**Les Cahiers du GERAD** ISSN: 0711–2440

# Post-separation feature reduction

J. W. Chinneck

G–2019–31

May 2019



**GERAD** HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

**Tél. : 514 340-6053**
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

# Post-separation feature reduction

**John W. Chinneck** [a,b]

[a] *GERAD, Montréal (Québec), Canada, H3T 2A7*

[b] *Systems and Computer Engineering, Carleton University, Ottawa (Ontario) Canada, K1S 5B6*

chinneck@sce.carleton.ca

**May 2019**
**Les Cahiers du GERAD**
**G–2019–31**

**Abstract:** Reducing the number of features used in data classification can remove noisy or redundant features, reduce the cost of data collection, and improve the accuracy of the classifier. This important step is normally conducted *before* a data separation (typically a hyperplane) is found. We reverse the process: a separating hyperplane is found first, and then *afterwards* a revised hyperplane is calculated that has fewer features, but that provides the same separation or better. The method relies on recent algorithms for finding sparse solutions for linear programs. Experiments show that the number of features in the separating hyperplane can be reduced substantially, while the overall accuracy often increases (but never reduces). This approach allows feature reduction to be easily incorporated into classifier decision tree construction using any algorithm for hyperplane selection. Separations that are substantially similar to the original separation but that use even fewer features can also be found. If the features have costs, the algorithm is easily extended to find separations that are at least as accurate as the original separation at much lower total cost.

**Keywords:** Classification, feature selection, maximum feasible subset, sparse solutions, linear programming

**Acknowledgments:** The Natural Sciences and Engineering Research Council of Canada (NSERC) provided funding to support this research.

# 1 Introduction

Feature selection for classification is a well-studied topic. Several excellent reviews are available, e.g. Li et al. (2018), Khalid et al. (2014) and Tang et al. (2014). Broadly speaking, feature selection methods are of three types: (i) filter methods apply metrics based on the data values to select features, (ii) wrapper methods select subsets of features and apply the classification technique to evaluate the selection, and (iii) hybrid methods combine elements of both filter and wrapper methods. In all these methods, the feature selection occurs *before* the classifier surface (e.g. hyperplane) is placed. A few methods (e.g. Chinneck (2012), Bradley et al. (1998), Maldonado et al. (2011)) select features during the hyperplane placement process.

This paper reverses the usual sequence: first find a separating hyperplane using any algorithm, and *then* find an equivalent or better separation using fewer features. Two key ideas are used: converting a set of data points into a set of linear inequalities for which a linear programming solution returns a separating hyperplane, and recent developments in finding *sparse solutions* for sets of linear constraints, i.e. solutions in which few of the variables take nonzero values.

The main steps are: (i) find a separating hyperplane by any method, (ii) identify the subset of points correctly classified by the hyperplane, (iii) convert the correctly classified points into a set of linear inequalities, and (iv) find a sparse solution for this set of linear inequalities. The sparse solution provides a separating hyperplane that uses fewer features than the original hyperplane. In our experiments, the number of features used is almost always reduced, frequently substantially, and the total accuracy of the separation is frequently increased.

The advantage of this approach is that any effective hyperplane placement method can be used with feature reduction applied afterwards. There are also options to consider the cost of the features or to limit the number of features to include.

The contributions are: (i) a method to reduce the number of features used in any separating hyperplane while often boosting the accuracy, (ii) a method to further reduce the number of features used while obtaining a similar (but not identical) separation with at least the same accuracy, and (iii) a method to heuristically minimize the total cost of the features, where features have costs, while maintaining the same or a similar separation. Relevant background follows on the maximum feasible subset problem and its application in finding separating hyperplanes and finding sparse solutions for linear programs.

## 1.1 The Maximum Feasible Subset Problem (maxFS)

The *Maximum Feasible Subset Problem* (*maxFS*) is this: given a set of constraints, find the largest cardinality feasible subset (Amaldi et al. 1999) (Chinneck 2007, ch. 7). This problem occurs in many applications, such as training neural networks, finding the data depth, radiation treatment planning, protein folding prediction, etc. (Chinneck 2007, 2019). It also occurs in finding separating hyperplanes for classification, as described in Section 1.2.

*MaxFS* is NP-hard (Sankaran 1993), but there are high quality heuristics in the case of linear constraints. Chinneck's (1996, 2001) algorithms for sets of linear constraints are generally the most effective. They first convert the constraints to an *elastic* format by including nonnegative elastic variables $e_i$ that permit the constraints to be violated:

| *nonelastic constraint i* | *elastic version* |
|---|---|
| $\boldsymbol{a}_i\boldsymbol{x} \geq b_i$ | $\boldsymbol{a}_i\boldsymbol{x} + e_i \geq b_i$ |
| $\boldsymbol{a}_i\boldsymbol{x} \leq b_i$ | $\boldsymbol{a}_i\boldsymbol{x} - e_i \leq b_i$ |
| $\boldsymbol{a}_i\boldsymbol{x} = b_i$ | $\boldsymbol{a}_i\boldsymbol{x} + e_i' - e_i''$ |

All constraints are elasticized and a linear program (LP) having the elastic objective function $min\ Z = \sum_i (e_i + e_i^{'} + e_i^{"})$ is solved. This minimizes the sum of the constraint violations at the solution point.

Chinneck's algorithms operate in an iterative fashion, at each iteration removing the constraint that most reduces the elastic objective function value when temporarily deleted. The reduced set of constraints is feasible when $Z = 0$, and the retained constraints constitute the heuristic maximum feasible subset. The number of linear programs that must be solved is reduced by limiting the set of constraints that are candidates for removal testing in each iteration. Note that each LP differs from the previous one by at most two constraints, so the LP solver takes advantage of advanced starts, so the process is relatively quick.

An overview of Chinneck's *maxFS* solution algorithm for sets of linear constraints is given in Algorithm 1. $N_{viol}$ is the number of violated constraints at the current LP solution point. There are several ways to select the set of candidate constraints following an LP solution. The original method includes all constraints whose elastic variables have a nonzero reduced cost. A second method chooses only constraints having nonzero elastic variables. These two methods can also be combined. A third method estimates the change in the elastic objective function value by calculating the product of the value of the elastic variable and the shadow price of the constraint it appears in. Shorter lists can be constructed by sorting the lists in descending order by the absolute magnitudes of the measures and taking just the top $k$ from the sorted list. See (Chinneck 2007) for further details.

**Algorithm 1** Chinneck (1996) *maxFS* heuristic

**INPUT**: Set of constraints $C$ defining an infeasible system of linear constraints.
0. Set up the elastic LP.
1. Solve the elastic LP.
   **IF** $N_{viol} = 1$ **THEN** $C \leftarrow C \backslash \{$violated constraint$\}$ and exit.
   $NextSet \leftarrow \{$candidate constraints from LP solution$\}$.
2. $Z_{min} \leftarrow \infty$.
   $CandidateSet \leftarrow NextSet$.
   **FOR** each constraint in $CandidateSet$:
      Delete the constraint.
      Solve elastic LP to find $Z_{elastic}$.
      **IF** $Z_{elastic} = 0$ **THEN** $C \leftarrow C \backslash \{$current constraint$\}$ and exit.
      **IF** $Z_{elastic} < Z_{min}$ **THEN**:
         $Winner \leftarrow$ current constraint.
         $Z_{min} \leftarrow Z_{elastic}$.
         $NextSet \leftarrow \{$candidate constraints from LP solution$\}$.
         **IF** $N_{viol} = 1$ **THEN** $NextWinner \leftarrow$ single violated constraint.
         **ELSE** $NextWinner \leftarrow \emptyset$.
      Reinstate the constraint.
3. $C \leftarrow C \backslash \{Winner\}$.
   **IF** $NextWinner \neq \emptyset$ **THEN** $C \leftarrow C \backslash \{NextWinner\}$ and exit.
   Go to Step 2.
**OUTPUT**: $C$ is a small set of constraints constituting a heuristic *maxFS* solution.

## 1.2 Linear classifiers and maxFS

In a series of papers, Chinneck (2001, 2009, 2012) showed how *maxFS* solution heuristics can be used to find separating hyperplanes of very high accuracy, and can also be used to pursue such classification objectives as preventing null hyperplanes when there are few minority data points relative to the majority type, balancing the prediction accuracy of each type, favouring the prediction accuracy of a given type, etc. In all cases, the first step is the conversion of the binary classification problem into a *maxFS* problem as described next.

Given a training set of $I$ data points ($i$=1...$I$) in $J$ dimensions ($j$=1...$J$), in which the value of attribute $j$ for point $i$ is denoted by $d_{ij}$, and the class of each point is known (either type 0 or type 1),

define a set of linear inequality constraints as follows (one constraint for each data point):

$$\begin{gathered}\text{Type 0 point} : \sum_j d_{ij} w_j \leq w_0 - \epsilon \\ \text{Type 1 point} : \sum_j d_{ij} w_j \geq w_0 + \epsilon \\ \text{equivalently :} \\ \text{Type 0 point} : -\sum_j d_{ij} w_j + w_0 \geq \epsilon \\ \text{Type 1 point} : \sum_j d_{ij} w_j - w_0 \geq \epsilon\end{gathered} \tag{1}$$

where $\epsilon$ is a small positive constant (normally set at 1), the variables are the unrestricted $w_j$, where $j$=0... $J$, and the $d_{ij}$ are known constants. The *maxFS* solution for this problem finds the separating hyperplane $\Sigma_j d_{ij} w_j = w_0$ that heuristically classifies the largest number of data points correctly. New points are tested by calculating $\Sigma_j d_{ij} w_j - w_0$: points having negative values are deemed to be of type 0 and points having positive values are deemed to be of type 1. The value of $\epsilon$ affects the scaling of $w$ and $w_0$.

We make extensive use of this transformation in the sequel.

Chinneck (2012) uses a *maxFS* approach to simultaneous hyperplane placement and feature selection by adding feature zeroing constraints of the form $w_j = 0$ to the data point constraints given in Equation (1). Now at each iteration Algorithm 1 can either eliminate a data point constraint (thereby misclassifying the data point) or eliminate a feature zeroing constraint (thereby introducing the feature), whichever most reduces $Z_{elastic}$. This is very effective in the reported experiments.

## 1.3 Sparse solutions for sets of linear constraints

In a sparse solution for a set of linear constraints, only a small number of the variables take nonzero values. Effective *maxFS*-based methods for solving this problem are now available (Chinneck 2018). One approach is to add variable zeroing constraints of the form $x_j = 0$ and then find a maximum feasible subset of those constraints while honouring all the other constraints.

A more parsimonious *maxFS* formulation is due to Jokar and Pfetsch (2008) for finding sparse solutions to underdetermined sets of linear equations in unrestricted variables. They set up a linear program to minimize the sum of the variable magnitudes. Because the variables can take positive or negative values, a change of variables is necessary: each variable $x_j$ is replaced by the difference of two nonnegative variables $u_j$ and $v_j$, *i.e.* $x_j = u_j - v_j$. We extend this to include inequalities, so the initial LP is:

$$\begin{gathered}\min Z = \sum_j (u_j + v_j) \\ \text{s.t.} \sum_j a_{ij}(u_j - v_j)\{\leq, \geq, =\} b_i, \text{ for } i = 1 \ldots m \\ u_j \geq 0, v_j \geq 0, \text{ for } j = 1 \ldots n.\end{gathered} \tag{2}$$

The elasticity is implicit in the magnitudes of the variables in that they "stretch" away from zero against the "pressure" of the objective forcing them towards zero. The *maxFS* algorithm is modified to add variables to the *support* (the set of nonzero variables in the solution) rather than removing them. Variables (or more accurately the substituted *u*-*v* pairs) are added to the support by removing them from the objective function, thus removing the penalty associated with their nonzero value. When enough variables have been removed from the objective function, the value of $Z$ is reduced to zero and the algorithm terminates. The variables removed from the objective function are the nonzeros in the sparse solution; the variables still in the objective function all take zero values.

It is not unusual for the base algorithm to include unnecessary nonzero variables. These can often be removed by a postprocessing step: each variable is removed in turn from the support, and the feasibility of the constraint set is tested; if the set is infeasible, then the variable is returned to the support, otherwise it is dropped permanently. The method is summarized in Algorithm 2.

**Algorithm 2** Heuristic for finding a sparse solution for a set of linear constraints.

**INPUT**: a set of linear constraints in variables j = 1... n
0. Set up the initial LP (Equation 2).
1. Solve the current LP.
2. $minZ \leftarrow \infty$. $Winner \leftarrow 0$.
3. For all $u$-$v$ pairs having nonzero objective coefficients and nonzero $|u - v|$: sort in descending order of $|u - v|$.
4. **IF** the list is empty, **THEN** go to $PostProcess$.
5. **FOR** the first $k$ $u$-$v$ variable pairs in the list:
   5.1. Set the objective coefficients of the $u$-$v$ pair to zero.
   5.2. Solve the revised LP.
   5.3. **IF** $Z = 0$, then go to $PostProcess$.
   5.4. **IF** $Z < minZ$:
      5.4.1. $minZ \leftarrow$ Z.
      5.4.2. $Winner \leftarrow$ current $u$-$v$ pair.
   5.5. Reset the objective function values of the current $u$-$v$ pair to 1.
6. Set the objective function coefficients of the $Winner$ $u$-$v$ pair to zero permanently.
7. Go to Step 1.

$PostProcess$:
1. Remove all variables having nonzero objective coefficients.
2. For each $u$-$v$ pair in the reduced LP:
   2.1. Remove the $u$-$v$ pair from the LP.
   2.2. Solve the LP.
   2.3. **IF** the LP is infeasible, return the $u$-$v$ pair to the LP.

**OUTPUT**: the set of $u$-$v$ pairs remaining in the LP identifies the support.

The candidate list can be adjusted to any length $k$. This speeds the solution process, often with little or no impact on solution quality.

Chinneck (2018) extends Algorithm 2 to all forms of LPs, including inequalities, ranged constraints, and bounded variables. Modifications include:

- Eliminate from the objective function all variables that must be in the support due to their bounds (e.g. $x_j \geq 1$) and ignore them when establishing candidate lists.
- Handle variables that are restricted to nonpositivity by setting their objective coefficients to $-1$.
- Handling variables whose span crosses zero via the usual change of variables.

Otherwise the algorithm is essentially as in Algorithm 2 and is referred to here as Algorithm 3.

# 2 Feature reduction and sparse LP solutions

The general approach developed here is flexible. It can be used to find an equivalent or better separation using fewer features (Section 2.1), to find a similar separation using a limited number of features, and to heuristically minimize the total feature cost.

## 2.1 Equivalent separation with fewer features

Given a separating hyperplane found by any method, the algorithms developed here find a new separating hyperplane equation that correctly classifies the same subset of points while using fewer features. The main steps of the algorithm are:

1. Given the initial separating hyperplane, identify the subset of points that it classifies correctly.
2. Convert the correctly classified points into a set of linear inequalities (Equation (1)).

3. Find a sparse solution for this feasible set of linear inequalities using Algorithm 3.
4. Given the sparse set of retained features, find the sparse separating hyperplane.

Step 4 solves a single LP in the reduced set of constraints from Step 2 and the reduced set of variables from Step 3. The LP is a version of Equation (2) in which the $n$ feature variables $w_j$ are converted by the change of variables $w_j = u_j - v_j$:

$$\begin{gathered} \min Z = \sum_j (u_j + v_j) \\ \text{s.t.} \\ \forall \text{ type 0 points: } -\sum_j d_{ij}(u_j - v_j) + w_0 \geq \epsilon \\ \forall \text{ type 1 points: } \sum_j d_{ij}(u_j - v_j) - w_0 \geq \epsilon \\ u_j \geq 0,\ v_j \geq 0, \text{ for } j = 1 \ldots n.\ w_0 \text{ unrestricted.} \end{gathered} \tag{3}$$

The final sparse separating hyperplane is recovered by reversing the change of variables to recover the $w_j$ for $j$=1…$n$, while $w_0$ is provided directly by the LP solution.

This is the procedure followed in the experiments conducted in Section 3.

## 2.2 Limiting the number of features

Suppose you wish to limit the number of features to be no more than some upper value $f_{max}$: which features should you choose, and how should you select their weights? A clue lies in the fact that the *maxFS* heuristic identifies the features to add in a specific order from most to least important. At each iteration the feature that most reduces the total magnitude of the feature weights is selected. Thus, the first $f_{max}$ features selected are the $f_{max}$ most important. Selecting the first $f_{max}$ features, it remains only to determine their weights in the separating hyperplane.

The procedure is as follows:

1. Given the initial separating hyperplane, identify the subset of points that it classifies correctly.
2. Convert the correctly classified points into a set of linear inequalities (Equation (1)).
3. Apply Algorithm 3 until $f_{max}$ features have been selected (or fewer if the algorithm terminates before $f_{max}$ features have been selected).
4. Construct a classification LP using the constraints from Step 2, and the subset of selected features from Step 3. Elasticize each constraint and solve the LP to minimize the sum of the elastic variables. The values of $w$ and $w_0$ are given by the LP solution. The LP has the form:

$$\begin{gathered} \min Z = \sum_i e_i \\ \text{s.t.} \\ \forall \text{ type 0 points: } -\sum_j d_{ij} w_j + w_0 + e_i \geq \epsilon \\ \forall \text{ type 1 points: } \sum_j d_{ij} w_j - w_0 + e_i \geq \epsilon \\ w_0, w_j \text{ for } j = 1 \ldots n \text{ unrestricted. } e_i \geq 0 \text{ for } i = 1 \ldots m. \end{gathered} \tag{4}$$

Assuming that $f_{max}$ features are insufficient to correctly classify all of the points that the original hyperplane correctly satisfied, the rationale for this method is that minimizing the total error, as measured by the sum of the constraint elastic variables, will yield an acceptably useful separating hyperplane.

## 2.3 The incremental method

A separating hyperplane is found at each LP solution in Algorithm 3. As shown experimentally later, it frequently happens that the separating hyperplane at some intermediate iteration has a total accuracy at least as high as for the original hyperplane. This underlies the *incremental method*, which modifies Algorithm 3 to exit as soon as such a hyperplane is identified at some intermediate LP solution. This speeds the solution as well as using fewer features. Note that the returned hyperplane does not guarantee to correctly classify all the points that were correctly classified by the original hyperplane. However, the incremental method is always working towards correctly classifying those points, so the separating hyperplane provides a separation that is substantially similar.

## 2.4 Minimizing the feature costs

If there is a cost associated with obtaining feature data (e.g. costs associated with an X-ray or other medical procedure), then instead of trying to minimize the *number* of features used to produce an equivalent separation, one may wish to minimize the total feature *costs* while finding an equivalent separation. This can be formulated as a binary integer programming (BIP) problem as follows.

Using the second form of the inequalities in Equation (1) creates the initial constraint matrix $A$ in $n$+1 variables ($n$ feature weights plus $w_0$), expanded to $2n$+1 variables by the change of variables $w_j = u_j - v_j$ for the feature weights. Let $c_j$ represent the cost of feature $j$, and $y_j$ be a binary variable indicating whether feature $j$ is to be used or not. $M$ is the usual large positive constant ("big $M$"). The BIP formulation is then:

$$\begin{gathered}
\min Z = \sum_j c_j y_j \\
\text{s.t.} \\
\forall \text{ type 0 points: } -\sum_j d_{ij}(u_j - v_j) + w_0 \geq \epsilon \\
\forall \text{ type 1 points: } \sum_j d_{ij}(u_j - v_j) - w_0 \geq \epsilon \\
u_j + v_j - M y_j \leq 0, \text{ for } j = 1 \ldots n \\
u_j \geq 0, \; v_j \geq 0, \text{ for } j = 1 \ldots n. \, w_0 \text{ unrestricted.} \\
y_j \in \{0, 1\} \text{ for } j = 1 \ldots n.
\end{gathered} \tag{5}$$

The first two sets of constraints represent the converted correctly classified data point constraints. The third set of constraints insures that $y_j$=1 if feature $j$ is used. Where the original data set has $m$ instances in $n$ features, the BIP formulation has $m$+$n$ constraints in $2n$+1 continuous and $n$ binary variables.

BIP formulations of this type are sensitive to the value of $M$. It must be larger than the largest $(u_j + v_j)$ for any $j$, but not so large that it causes numerical difficulties. In practice, to keep the sizes of $(u_j + v_j)$ small, it is necessary to give each $u_j$ and $v_j$ a very small objective coefficient, e.g. on the order of 0.0001, where $M$ is e.g. 10000. In our experiments, the BIP solution often failed, stymied by the range of values of the features. For example, the *kc2* dataset has a feature whose range is 0 to 2,147,483.64. There is therefore still a need for a heuristic method. Fortunately, it is straightforward to adjust the *maxFS* heuristic to consider feature costs.

The cost-minimizing version of *maxFS* uses a revised version of Algorithm 3. The difference is in the selection and evaluation of candidates. All candidates must have a nonzero magnitude; they are then sorted in ascending order by (feature cost)/(feature variable magnitude) so that the candidates having the lowest cost per magnitude are selected first if the list length is limited. At an intermediate LP solution, the feature variables are in three sets: (i) those already selected, whose objective values have been permanently set to zero, (ii) the current candidate feature variable, whose objective value

is temporarily set to zero, and (iii) the remaining feature variables, some of which have positive magnitudes in the current LP solution, and some of which are zero. All candidates in the current round have set (i) in common, so the metric is (feature cost for current candidate feature) + (feature costs for all features in set (iii) that have positive magnitudes). This metric tries to estimate the remaining total feature costs incurred by selecting the current candidate. The winning candidate has the smallest such value.

The algorithm exits when enough features have been added so that the hyperplane accuracy meets or exceeds the original hyperplane accuracy, as in the incremental method of Section 2.3. As for the incremental method this implies that the separation is substantially similar to the original separation but does not necessarily correctly classify all of the data points correctly classified by the original.

# 3 Experiments

Four sets of experiment were conducted. Experiment 1 evaluates the ability to provide an equivalent separation using fewer features and possibly with higher accuracy (method of Section 2.1). Experiment 2 evaluates the impact of an *a priori* limit on the number of features allowed. Experiment 3 demonstrates the reduced features and improved speed due to the incremental method of Section 2.3. Experiment 4 evaluates the ability to find a low-cost set of features that provides a similar separation where the features have costs.

## 3.1 Experimental setup

**Software.** All experiments were conducted in Matlab R2017b running in Windows 10 (Matlab 2019). The linear programming solver was the MOSEK Optimization Toolbox for MATLAB Release 8.1.0.34 (Mosek 2019).

**Hardware.** Intel Xeon CPU E5-1650 operating at 3.50 GHz, 6 cores with 12 logical processors, 32 Gbytes RAM. The LP solver uses a single logical processor.

**Datasets.** The 28 datasets were chosen as a mix of relatively small but well-studied datasets and larger datasets with more features. Instances having missing values were removed. Dataset sizes range from 600 to 3,363,960 nonzeros, from 150 to 3,468 instances, and from 4 to 10,936 features. All are binary classifications with numerical features only. Their basic statistics are summarized in Table 1.

**Metrics.** The number of features and the total accuracy of the original and sparse hyperplanes are compared. The time taken to find the sparse hyperplane is reported. A feature is counted if its absolute weight is at least 1.0e-6. This is an accurate measure for the sparse hyperplanes due to the final step which finds the hyperplane weights while including only the identified columns, but it can be an underestimate for the original hyperplanes which may not use the same offset and hence hyperplane scaling. For reproducibility, the original hyperplane placement and feature reduction methods are always applied to the entire dataset.

## 3.2 Experiment 1: Find an equivalent separation using fewer features

This experiment evaluates the ability of the *maxFS*-based method of Section 2.1 to reduce the number of features used to produce a separation that is equivalent to (or better than) the separation produced by any given hyperplane placement method. Three original hyperplane placement methods are used:

- An implementation of the *maxFS*-based hyperplane placement method (Chinneck 2001, 2009), with list length 1.
- The Matlab *fitclinear* function for support vector machine hyperplane placement with default parameters. Can give different results on different runs on the same dataset.

- The Matlab *fitclinear* function for logistic regression hyperplane placement with default parameters. Can give different results on different runs on the same dataset.

**Table 1: Dataset statistics.**

| | | | | | type 0 | | type 1 | |
|---|---|---|---|---|---|---|---|---|
| **Dataset** | **Src** | **instances** | **features** | **nonzeros** | **num** | **frac** | **num** | **frac** |
| AP_Lung_Uterus | 2 | 250 | 10936 | 2734000 | 124 | 0.496 | 126 | 0.504 |
| bodyfat | 2 | 252 | 14 | 3528 | 128 | 0.508 | 124 | 0.492 |
| breast1: breast cancer Wisconsin (original) | 1 | 683 | 9 | 6147 | 444 | 0.650 | 239 | 0.350 |
| bupa: liver disorders | 1 | 345 | 6 | 2070 | 145 | 0.420 | 200 | 0.580 |
| collins | 2 | 500 | 22 | 11000 | 420 | 0.840 | 80 | 0.160 |
| fri_c4_500_100 | 2 | 500 | 100 | 50000 | 283 | 0.566 | 217 | 0.434 |
| gina_agnostic | 2 | 3468 | 970 | 3363960 | 1763 | 0.508 | 1705 | 0.492 |
| glass: type 2 vs. others | 1 | 214 | 9 | 1926 | 138 | 0.645 | 76 | 0.355 |
| hillvalley | 2 | 1212 | 100 | 121200 | 600 | 0.495 | 612 | 0.505 |
| ionosphere | 1 | 351 | 34 | 11934 | 126 | 0.359 | 225 | 0.641 |
| iris: versicolor vs. others | 1 | 150 | 4 | 600 | 100 | 0.667 | 50 | 0.333 |
| jasmine1 | 2 | 2984 | 144 | 429696 | 1492 | 0.500 | 1492 | 0.500 |
| kc2 | 2 | 522 | 21 | 10962 | 415 | 0.795 | 107 | 0.205 |
| MegaWatt1 | 2 | 253 | 37 | 9361 | 226 | 0.893 | 27 | 0.107 |
| musk1: musk version 1 | 1 | 476 | 166 | 79016 | 269 | 0.565 | 207 | 0.435 |
| newthyroid: type 1 vs. others | 1 | 215 | 5 | 1075 | 65 | 0.302 | 150 | 0.698 |
| oil_spill | 2 | 937 | 49 | 45913 | 896 | 0.956 | 41 | 0.044 |
| pima: Pima Indians Diabetes | 1 | 768 | 8 | 6144 | 500 | 0.651 | 268 | 0.349 |
| scene | 2 | 2407 | 294 | 707658 | 1976 | 0.821 | 431 | 0.179 |
| segmentation: image segmentation | 1 | 2310 | 19 | 43890 | 1980 | 0.857 | 330 | 0.143 |
| sonar: Connectionist Bench (Sonar, Mines vs. Rocks) | 1 | 208 | 60 | 12480 | 97 | 0.466 | 111 | 0.534 |
| SPECTF | 2 | 349 | 44 | 15356 | 95 | 0.272 | 254 | 0.728 |
| tokyo1 | 2 | 959 | 44 | 42196 | 346 | 0.361 | 613 | 0.639 |
| vehicle: Statlog (Vehicle Silhouettes) bus vs.others | 1 | 850 | 18 | 15300 | 632 | 0.744 | 218 | 0.256 |
| wdbc: breast cancer Wisconsin (diagnostic) | 1 | 569 | 30 | 17070 | 357 | 0.627 | 212 | 0.373 |
| wine: type 3 vs. others | 1 | 178 | 13 | 2314 | 130 | 0.730 | 48 | 0.270 |
| wpbc: Breast Cancer Wisconsin (Prognostic) | 1 | 194 | 32 | 6208 | 166 | 0.856 | 28 | 0.144 |
| yeast | 2 | 2417 | 103 | 248951 | 1655 | 0.685 | 762 | 0.315 |

Sources: 1 – Dua and Graff (2019), 2 – OpenML (2019). Note: Pima dataset later removed from source 1 repository.

The list length for the *maxFS*-based feature reduction algorithm is 7. Results are summarized in Table 2. *Best accuracy* indicates the number of models for which the original hyperplane placement method has the highest total accuracy, of the 28 tested data sets (there is a tie in one case). *Time outs* gives the number of data sets for which the original placement method exceeded the time limit of 1 hour. *MaxFS reduces features* and *maxFS increases accuracy* gives the number of data sets having these outcomes.

**Table 2: Summary results.**

| **Original Placement Method** | **Best accuracy** | **Time outs** | **maxFS reduces features** | **maxFS increases accuracy** | **Average fractional reduction in number of features** |
|---|---|---|---|---|---|
| *maxFS*-based LL1 | 27 | 1 | 20 | 2 | 0.278 |
| Matlab SVM | 2 | 0 | 27 | 22 | 0.675 |
| Matlab logistic | 0 | 0 | 25 | 23 | 0.661 |

The final column shows that the *maxFS* sparse features algorithm is highly successful in reducing the number of features, removing up to 67.5% of the features used by the original hyperplane, on average, and much higher fractions in many cases, as shown in the following tables. Note that significantly fewer features are removed for the higher-accuracy *maxFS*-based original hyperplane placement. The

higher accuracy arises from the effective use of more features, so fewer features can be removed while finding a sparse hyperplane that maintains the same separation.

Table 3 to Table 5 summarize the specific results for the 3 original hyperplane placement methods. *NF* indicates the number of features used in the separating hyperplane; *Acc* is the hyperplane total accuracy. The smaller number of features and the higher accuracy between the original and the sparse hyperplanes are shown in boldface. *Redn* is the reduction in the number of features in the sparse hyperplane versus the original hyperplane; *fRedn* is the fractional reduction in the number of features.

**Table 3: Original hyperplane: maxFS-based LL1 with average accuracy 94.4%.**

| | Original | | Sparse | | | | |
|---|---|---|---|---|---|---|---|
| **Dataset** | **NF** | **Acc** | **NF** | **Acc** | **Time** | **Redn** | **fRedn** |
| AP_Lung_Uterus | 117 | 1.000 | **9** | 1.000 | 116.7 | 108 | 0.923 |
| bodyfat | 10 | 1.000 | **1** | 1.000 | 0.0 | 9 | 0.900 |
| Breast1 | 9 | 0.984 | 9 | 0.984 | 0.2 | 0 | 0.000 |
| bupa | 6 | 0.759 | 6 | 0.759 | 0.1 | 0 | 0.000 |
| collins | 20 | 0.992 | **15** | 0.992 | 0.9 | 5 | 0.250 |
| fri_c4_500_100 | 100 | 0.924 | **82** | 0.924 | 106.3 | 18 | 0.180 |
| gina_agnostic | *timeout* | | | | | | |
| glass | 9 | 0.790 | 9 | 0.790 | 0.2 | 0 | 0.000 |
| hillvalley | 6 | 1.000 | **3** | 1.000 | 1.6 | 3 | 0.500 |
| ionosphere | 33 | 0.983 | **22** | 0.983 | 1.7 | 11 | 0.333 |
| iris | 4 | 0.833 | 4 | 0.833 | 0.0 | 0 | 0.000 |
| jasmine1 | 137 | 0.884 | **123** | 0.884 | 247.6 | 14 | 0.102 |
| kc2 | 21 | 0.891 | **18** | 0.891 | 0.7 | 3 | 0.143 |
| MegaWatt1 | 32 | 0.972 | **24** | 0.972 | 1.5 | 8 | 0.250 |
| musk1 | 166 | 1.000 | **51** | 1.000 | 119.5 | 115 | 0.693 |
| newthyroid | 5 | 0.949 | 5 | 0.949 | 0.0 | 0 | 0.000 |
| oil_spill | 46 | 0.995 | **35** | **0.996** | 8.0 | 11 | 0.239 |
| pima | 8 | 0.801 | 8 | 0.801 | 0.2 | 0 | 0.000 |
| scene | 294 | 0.982 | **279** | 0.982 | 9771.1 | 15 | 0.051 |
| segmentation | 14 | 1.000 | **7** | 1.000 | 1.6 | 7 | 0.500 |
| sonar | 60 | 1.000 | **35** | 1.000 | 5.5 | 25 | 0.417 |
| SPECTF | 44 | 0.977 | **32** | 0.977 | 3.7 | 12 | 0.273 |
| tokyo1 | 40 | 0.967 | **23** | **0.968** | 4.4 | 17 | 0.425 |
| vehicle | 16 | 0.987 | **13** | 0.987 | 0.9 | 3 | 0.188 |
| wdbc | 28 | 1.000 | **24** | 1.000 | 2.0 | 4 | 0.143 |
| wine | 13 | 1.000 | **3** | 1.000 | 0.0 | 10 | 0.769 |
| wpbc | 32 | 0.969 | **25** | 0.969 | 1.3 | 7 | 0.219 |
| yeast | 103 | 0.862 | 103 | 0.862 | 287.3 | 0 | 0.000 |

The tables show that in the vast majority of cases, the sparse hyperplane uses fewer features and very often has a higher total accuracy, especially for the lower-accuracy original hyperplane placement methods. The accuracy of the sparse hyperplane is at least as high as for the original hyperplane because it is required to correctly classify the same set of points as correctly classified by the original hyperplane. However, the sparse hyperplane may correctly classify additional points, and hence may have higher total accuracy.

Table 5 shows two anomalies: for the *hillvalley* and *tokyo1* datasets, the original Matlab logistic regression hyperplane uses no features while the sparse hyperplane uses 1. This appears to be due to the scaling of the hyperplane returned by the Matlab logistic regression. For *hillvalley*, all elements of $w$ are on the order of 1.0e-8 with $w_0 = -0.0099$; scaling up by a factor of 100 would result in all elements of $\boldsymbol{w}$ surpassing the inclusion threshold. For *tokyo1*, 5 elements of $\boldsymbol{w}$ are on the order of 1.0e-8 or 1.0e-7 with $w_0 = -0.2784$, so scaling up by a factor of 100 would result in 5 features passing the inclusion threshold.

The sparse hyperplane solution times are relatively small for 21 of the datasets but can be larger for 7 of them: *AP_Lung_Uterus, fri_c4_500_100, gina_agnostic, jasmine1, musk1, scene,* and *yeast* have solution times over 60 seconds in at least one experiment. These are among the larger models in terms of total nonzeros and especially number of features. The sparse hyperplane solution time is also affected by the accuracy of the original hyperplane: a less accurate original hyperplane results in

**Table 4: Original hyperplane: Matlab SVM with average accuracy 86.2%.**

| | Original | | Sparse | | | | |
|---|---|---|---|---|---|---|---|
| **Dataset** | **NF** | **Acc** | **NF** | **Acc** | **Time** | **Redn** | **fRedn** |
| AP_Lung_Uterus | 50 | 0.988 | **6** | 0.988 | 74.9 | 44 | 0.880 |
| bodyfat | 14 | 0.857 | **8** | **0.881** | 0.2 | 6 | 0.429 |
| Breast1 | 9 | 0.974 | **5** | **0.980** | 0.4 | 4 | 0.444 |
| bupa | 6 | 0.687 | **5** | **0.699** | 0.1 | 1 | 0.167 |
| collins | 22 | 0.960 | **9** | **0.970** | 0.5 | 13 | 0.591 |
| fri_c4_500_100 | 100 | 0.784 | **34** | **0.824** | 26.7 | 66 | 0.660 |
| gina_agnostic | 968 | 0.896 | **115** | **0.932** | 6642.2 | 853 | 0.881 |
| glass | 9 | 0.645 | **0** | 0.645 | 0.0 | 9 | 1.000 |
| hillvalley | 100 | 1.000 | **3** | 1.000 | 1.7 | 97 | 0.970 |
| ionosphere | 33 | 0.923 | **10** | **0.934** | 0.6 | 23 | 0.697 |
| iris | 4 | 0.740 | **3** | **0.760** | 0.0 | 1 | 0.250 |
| jasmine1 | 144 | 0.776 | **28** | **0.805** | 16.8 | 116 | 0.806 |
| kc2 | 21 | 0.837 | **3** | 0.837 | 0.1 | 18 | 0.857 |
| MegaWatt1 | 37 | 0.885 | **0** | **0.893** | 0.0 | 37 | 1.000 |
| musk1 | 166 | 0.824 | **20** | **0.863** | 8.3 | 146 | 0.880 |
| newthyroid | 5 | 0.921 | 5 | **0.926** | 0.1 | 0 | 0.000 |
| oil_spill | 28 | 0.956 | **0** | 0.956 | 0.0 | 28 | 1.000 |
| pima | 8 | 0.688 | **6** | **0.691** | 0.2 | 2 | 0.250 |
| scene | 294 | 0.833 | **62** | **0.885** | 706.1 | 232 | 0.789 |
| segmentation | 19 | 0.997 | **3** | **0.998** | 0.6 | 16 | 0.842 |
| sonar | 60 | 0.841 | **10** | **0.875** | 0.7 | 50 | 0.833 |
| SPECTF | 44 | 0.828 | **11** | **0.868** | 0.9 | 33 | 0.750 |
| tokyo1 | 7 | 0.798 | **6** | **0.861** | 0.7 | 1 | 0.143 |
| vehicle | 18 | 0.942 | **11** | **0.965** | 0.7 | 7 | 0.389 |
| wdbc | 30 | 0.933 | **4** | **0.938** | 0.2 | 26 | 0.867 |
| wine | 13 | 0.972 | **2** | **0.989** | 0.0 | 11 | 0.846 |
| wpbc | 30 | 0.856 | **0** | 0.856 | 0.0 | 30 | 1.000 |
| yeast | 103 | 0.798 | **34** | **0.812** | 60.4 | 69 | 0.670 |

**Table 5: Original hyperplane: Matlab logistic regression with average accuracy 82.9%.**

| | Original | | Sparse | | | | |
|---|---|---|---|---|---|---|---|
| **Dataset** | **NF** | **Acc** | **NF** | **Acc** | **Time** | **Redn** | **fRedn** |
| AP_Lung_Uterus | 681 | 0.888 | **4** | **0.952** | 42.3 | 677 | 0.994 |
| bodyfat | 14 | 0.853 | **6** | **0.865** | 0.1 | 8 | 0.571 |
| Breast1 | 9 | 0.969 | **5** | **0.977** | 0.3 | 4 | 0.444 |
| bupa | 6 | 0.701 | **5** | 0.701 | 0.1 | 1 | 0.167 |
| collins | 22 | 0.962 | **9** | **0.974** | 0.6 | 13 | 0.591 |
| fri_c4_500_100 | 100 | 0.770 | **29** | **0.830** | 21.1 | 71 | 0.710 |
| gina_agnostic | 969 | 0.839 | **54** | **0.886** | 1340.4 | 915 | 0.944 |
| glass | 9 | 0.645 | **5** | **0.668** | 0.1 | 4 | 0.444 |
| hillvalley | **0** | 0.520 | 1 | **0.525** | 0.2 | *-1* | |
| ionosphere | 33 | 0.912 | **9** | **0.923** | 0.5 | 24 | 0.727 |
| iris | 4 | 0.727 | **3** | **0.740** | 0.0 | 1 | 0.250 |
| jasmine1 | 144 | 0.742 | **51** | **0.772** | 50.0 | 93 | 0.646 |
| kc2 | 2 | 0.837 | **1** | 0.837 | 0.0 | 1 | 0.500 |
| MegaWatt1 | 3 | 0.893 | **0** | 0.893 | 0.0 | 3 | 1.000 |
| musk1 | 166 | 0.765 | **12** | **0.798** | 3.8 | 154 | 0.928 |
| newthyroid | 5 | 0.907 | 5 | **0.926** | 0.1 | 0 | 0.000 |
| oil_spill | 1 | 0.956 | **0** | 0.956 | 0.0 | 1 | 1.000 |
| pima | 8 | 0.781 | **7** | **0.788** | 0.2 | 1 | 0.125 |
| scene | 294 | 0.842 | **31** | **0.873** | 202.2 | 263 | 0.895 |
| segmentation | 19 | 0.997 | **3** | **0.998** | 0.6 | 16 | 0.842 |
| sonar | 60 | 0.832 | **9** | **0.870** | 0.6 | 51 | 0.850 |
| SPECTF | 44 | 0.840 | **15** | **0.903** | 1.3 | 29 | 0.659 |
| tokyo1 | **0** | 0.511 | 1 | **0.522** | 0.1 | *-1* | |
| vehicle | 18 | 0.973 | **9** | **0.980** | 0.7 | 9 | 0.500 |
| wdbc | 30 | 0.935 | **5** | **0.944** | 0.3 | 25 | 0.833 |
| wine | 13 | 0.983 | **2** | **0.989** | 0.0 | 11 | 0.846 |
| wpbc | 10 | 0.856 | **0** | 0.856 | 0.0 | 10 | 1.000 |
| yeast | 103 | 0.785 | **28** | **0.802** | 45.5 | 75 | 0.728 |

a smaller base LP for the sparse hyperplane placement. Sparse hyperplane solution time is roughly proportional to (nonzeros) × (original hyperplane accuracy) × (original hyperplane features). The first two terms give the scale of the sparse hyperplane solution LP and the final term is proportional to the number of iterations needed.

## 3.3 Experiment 2: Limiting the Number of Features

Experiment 2 examines the effect of limiting the number of features on the resulting total accuracy when the method of Section 2.2 is applied. For this experiment, the maximum number of features permitted $f_{max}$ is limited to half the number of features used in the sparse hyperplane, rounded up, i.e. $f_{max} = NF_{sparse}/2$.

Experiments were run using the two Matlab algorithms for original hyperplane placement. The results for the Matlab SVM original hyperplane placement are summarized in Table 6. Results for the logistic regression are similar and are summarized later. $\Delta Acc$ gives the difference in accuracy between the original hyperplane and the feature-limited hyperplane.

**Table 6: Limiting features to half the number in the sparse hyperplane. Original hyperplane placed by Matlab SVM.**

| | Original | | 1/2 Sparse | | |
|---|---|---|---|---|---|
| **Dataset** | **NF** | **Acc** | **NF** | **Acc** | **ΔAcc** |
| AP_Lung_Uterus | 50 | 0.988 | 3 | 0.940 | -0.048 |
| bodyfat | 14 | 0.857 | 4 | 0.841 | -0.016 |
| Breast1 | 9 | 0.974 | 3 | 0.960 | -0.013 |
| bupa | 6 | 0.687 | 3 | 0.655 | -0.032 |
| collins | 22 | 0.960 | 5 | 0.908 | -0.052 |
| fri_c4_500_100 | 100 | 0.784 | 17 | 0.738 | -0.046 |
| gina_agnostic | 968 | 0.896 | 58 | 0.879 | -0.017 |
| glass | 9 | 0.645 | 0 | 0.645 | 0.000 |
| hillvalley | 100 | 1.000 | 2 | 0.993 | -0.007 |
| ionosphere | 33 | 0.923 | 5 | 0.880 | -0.043 |
| iris | 4 | 0.740 | 2 | 0.747 | 0.007 |
| jasmine1 | 144 | 0.776 | 8 | 0.791 | 0.015 |
| kc2 | 21 | 0.837 | 2 | 0.845 | 0.008 |
| MegaWatt1 | 37 | 0.885 | 0 | 0.893 | 0.008 |
| musk1 | 166 | 0.824 | 10 | 0.773 | -0.050 |
| newthyroid | 5 | 0.921 | 3 | 0.907 | -0.014 |
| oil_spill | 28 | 0.956 | 0 | 0.956 | 0.000 |
| pima | 8 | 0.688 | 3 | 0.648 | -0.039 |
| scene | 294 | 0.833 | 31 | 0.843 | 0.010 |
| segmentation | 19 | 0.997 | 2 | 0.990 | -0.008 |
| sonar | 60 | 0.841 | 5 | 0.808 | -0.034 |
| SPECTF | 44 | 0.828 | 6 | 0.814 | -0.014 |
| tokyo1 | 7 | 0.798 | 2 | 0.875 | 0.077 |
| vehicle | 18 | 0.942 | 6 | 0.954 | 0.012 |
| wdbc | 30 | 0.933 | 2 | 0.947 | 0.014 |
| wine | 13 | 0.972 | 1 | 0.938 | -0.034 |
| wpbc | 30 | 0.856 | 0 | 0.856 | 0.000 |
| yeast | 103 | 0.798 | 17 | 0.790 | -0.008 |

Some loss in total accuracy compared to the original hyperplane is expected, but the loss is remarkably small on average (just 1.2 percentage points). In fact, increases in total accuracy of up to 7.7 percentage points (*tokyo1*) are seen. The largest total accuracy loss is 5.3 percentage points (*collins*). Accuracy losses are small even on datasets such as *newthyroid* for which the sparse hyperplane method (Section 2.1) was unable to eliminate any features. This demonstrates that *maxFS* is very effective in providing an importance ordering for the features.

Results are similar when the original hyperplane placement is provided by the Matlab logistic regression. The average total accuracy loss is 0.3 percentage points, the maximum loss is 4.8 percentage points (*collins*), and the largest accuracy gain is 10.4 percentage points (*AP_Lung_Uterus*).

Solution times were not recorded but can be expected to be about half the time taken for the full sparse hyperplane placement since there will be half the number of *maxFS* iterations. The final LP solution (Step 4 in Section 2.2) takes relatively little time.

## 3.4 Experiment 3: The Incremental Method

The incremental method (Section 2.3) tracks the total accuracy as features are added. It often happens that a solution that matches or exceeds the original hyperplane accuracy is found using significantly fewer features than the sparse hyperplane solution method of Section 2.1. The general trend is for accuracy to increase as more features are used, but because this is a heuristic solution, there can be fluctuations in the accuracy of the current hyperplane solution as features are added. This is illustrated in Figure 1 for *fri_c4_500_100* where the original hyperplane is placed by the Matlab SVM. The original hyperplane accuracy is first matched or exceeded with only 28 features instead of the 34 used by the sparse hyperplane method. There is no guarantee that the correctly classified points are the same points as correctly classified by the original hyperplane, but because the method is working towards correct classification of those points, the separation is substantially similar.

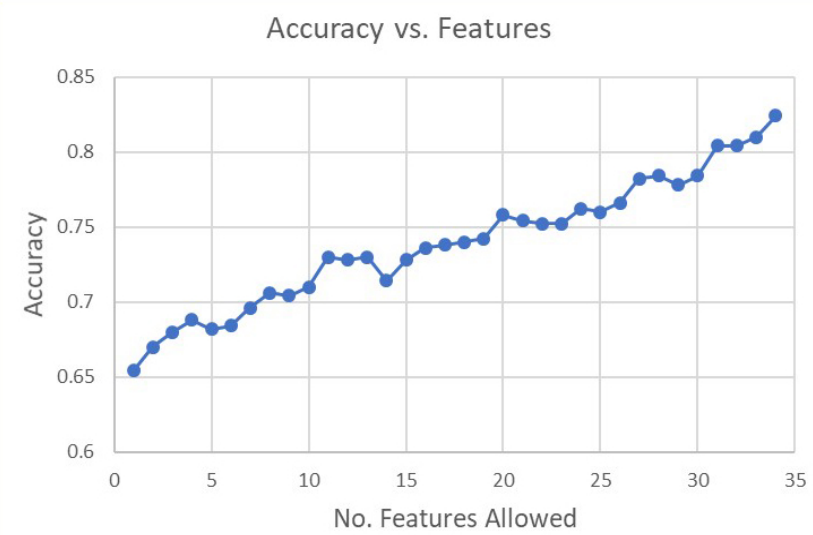


**Figure 1: Total accuracy vs permitted number of features for fri_c4_500_100.**

Results for the incremental method are given in Table 7, which compares the number of features and accuracy across (i) the original hyperplane found by the Matlab SVM, (ii) the sparse hyperplane via the method in Section 2.1, and (iii) the hyperplane found by the incremental method. Where there is a smaller number of features or higher accuracy between the sparse hyperplane and the incremental result, it is shown in boldface. The final cluster of columns *Incr – Sparse* compares the incremental and sparse hyperplane results. The incremental method uses fewer features than the sparse method in 16 of the 28 models, reducing the number of features by 21.3% on average (*fDiff NF*), and in some cases by much more, with only a minor loss in accuracy (1 percentage point on average). Incremental method total accuracy is never less than the original hyperplane accuracy (same in 7 cases, greater accuracy in 21 cases).

The incremental method also takes less time than the sparse method, with the time ratio compared to the sparse method about equal to the reduction in the number of features. This has a significant impact on the longer-running datasets such as *gina_agnostic*.

## 3.5 Experiment 4: Minimizing the Feature Costs

This experiment measures the ability of the algorithm of Section 2.4 to find a hyperplane that minimizes the total cost of the features while meeting or exceeding the accuracy of the original hyperplane.

Random feature costs between 0 and 100 are assigned, and results are compared to the cost associated with the original and the sparse hyperplanes, as well as the expected value of the cost. The logic is as follows:

1. Find the original hyperplane via Matlab SVM
2. Find the sparse hyperplane via Algorithm 3 with list length 7
3. Do 10 times:
   - 3.1 Assign random costs between 0 and 100 to each feature
   - 3.2 Calculate the cost for the original hyperplane
   - 3.3 Calculate the cost for the sparse hyperplane
   - 3.4 Apply the algorithm of Section 2.4 with list length 1 to find a low-cost hyperplane and its cost
4. Calculate means of costs for all three methods

Note that a list length of 1 is used in Step 3.4 to ensure that the candidate feature having the absolute lowest cost/magnitude is chosen.

**Table 7: Results for incremental method. Original hyperplane is Matlab SVM.**

| | Original | | Sparse | | Incremental | | | Incr - Sparse | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Dataset** | **NF** | **Acc** | **NF** | **Acc** | **NF** | **Acc** | **Time** | **NF** | **fDiff NF** | **Acc** |
| AP_Lung_Uterus | 44 | 0.984 | 6 | 0.988 | 6 | 0.988 | 68.3 | 0 | 0.000 | 0.000 |
| bodyfat | 14 | 0.857 | 8 | **0.881** | **6** | 0.861 | 0.1 | -2 | -0.250 | -0.020 |
| Breast1 | 9 | 0.974 | 5 | 0.980 | 5 | 0.980 | 0.3 | 0 | 0.000 | 0.000 |
| bupa | 6 | 0.687 | 5 | **0.699** | **4** | 0.696 | 0.1 | -1 | -0.200 | -0.003 |
| collins | 22 | 0.960 | 9 | 0.970 | 9 | **0.974** | 0.5 | 0 | 0.000 | 0.004 |
| fri_c4_500_100 | 100 | 0.784 | 34 | **0.824** | **28** | 0.784 | 24.5 | -6 | -0.176 | -0.040 |
| gina_agnostic | 967 | 0.880 | 106 | **0.918** | **59** | 0.883 | 4252.9 | -47 | -0.443 | -0.035 |
| glass | 9 | 0.645 | 0 | 0.645 | 0 | 0.645 | 0.0 | 0 | 0.000 | 0.000 |
| hillvalley | 100 | 1.000 | 3 | 1.000 | 3 | 1.000 | 1.5 | 0 | 0.000 | 0.000 |
| ionosphere | 33 | 0.923 | 10 | **0.934** | **9** | 0.926 | 0.6 | -1 | -0.100 | -0.009 |
| iris | 4 | 0.740 | 3 | 0.760 | 3 | **0.767** | 0.0 | 0 | 0.000 | 0.007 |
| jasmine1 | 144 | 0.785 | 31 | **0.813** | **7** | 0.795 | 11.3 | -24 | -0.774 | -0.019 |
| kc2 | 21 | 0.837 | 3 | 0.837 | **1** | **0.849** | 0.0 | -2 | -0.667 | 0.011 |
| MegaWatt1 | 37 | 0.885 | 0 | 0.893 | 0 | 0.893 | 0.0 | 0 | 0.000 | 0.000 |
| musk1 | 166 | 0.798 | 22 | **0.895** | **16** | 0.803 | 7.0 | -6 | -0.273 | -0.092 |
| newthyroid | 5 | 0.921 | 5 | 0.926 | 5 | 0.926 | 0.1 | 0 | 0.000 | 0.000 |
| oil_spill | 28 | 0.956 | 0 | 0.956 | 0 | 0.956 | 0.0 | 0 | 0.000 | 0.000 |
| pima | 8 | 0.688 | 6 | 0.691 | **4** | **0.721** | 0.1 | -2 | -0.333 | 0.030 |
| scene | 294 | 0.844 | 22 | **0.858** | **17** | 0.852 | 86.7 | -5 | -0.227 | -0.006 |
| segmentation | 19 | 0.997 | 3 | 0.998 | 3 | 0.998 | 0.7 | 0 | 0.000 | 0.000 |
| sonar | 60 | 0.841 | 10 | **0.875** | **8** | 0.856 | 0.6 | -2 | -0.200 | -0.019 |
| SPECTF | 44 | 0.828 | 11 | **0.868** | **9** | 0.828 | 0.7 | -2 | -0.182 | -0.040 |
| tokyo1 | 7 | 0.798 | 6 | **0.861** | **1** | 0.825 | 0.2 | -5 | -0.833 | -0.036 |
| vehicle | 18 | 0.942 | 11 | **0.965** | **6** | 0.954 | 0.6 | -5 | -0.455 | -0.011 |
| wdbc | 30 | 0.933 | 4 | 0.938 | **2** | **0.947** | 0.1 | -2 | -0.500 | 0.009 |
| wine | 13 | 0.972 | 2 | 0.989 | 2 | 0.989 | 0.0 | 0 | 0.000 | 0.000 |
| wpbc | 30 | 0.856 | 0 | 0.856 | 0 | 0.856 | 0.0 | 0 | 0.000 | 0.000 |
| yeast | 103 | 0.797 | 32 | **0.811** | **21** | 0.798 | 41.2 | -11 | -0.344 | -0.013 |
| *Average:* | *83.4* | *0.861* | *12.8* | *0.880* | *8.4* | *0.870* | *160.7* | *-4.4* | *-0.213* | *-0.010* |

The results are summarized in Table 8. Note that the number of features ($NF$) and total accuracy ($Acc$) are constant for the original and sparse hyperplanes over the 10 runs since they are unchanged when feature costs are changed. The hyperplane solution changes only for the incremental minimum cost method when the feature costs change. The number of features used, the accuracy, and the average costs are compared between the sparse solution and the incremental min-cost solution: if one result is better than the other then it is shown in boldface.

There are four cases where the average costs are equal because both methods return the null hyperplane. There are just 2 instances where the average costs for the sparse hyperplane are lower

**Table 8: Incremental cost minimization.**

| | Original | | | Sparse | | | Incremental Min-Cost | | | | | Cost Ratio to | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Dataset | NF | Acc | Avg Cost | NF | Acc | Avg Cost | Avg NF | Avg Acc | Avg Cost | Succ | Avg Time | Exp | Sparse |
| AP_Lung_Uterus | 51 | 0.984 | 2497.7 | **6** | **0.988** | 315.4 | 9.3 | 0.985 | **205.2** | 8 | 19.1 | 0.441 | 0.650 |
| bodyfat | 14 | 0.857 | 760.7 | 8 | **0.881** | 416.2 | **7.6** | 0.869 | **326.8** | 10 | 0.0 | 0.860 | 0.785 |
| Breast1 | 9 | 0.974 | 459.0 | 5 | **0.980** | 236.2 | 5 | 0.977 | **195.6** | 9 | 0.1 | 0.782 | 0.828 |
| bupa | 6 | 0.687 | 272.2 | 5 | **0.699** | 221.2 | **4.4** | 0.698 | **183.6** | 10 | 0.0 | 0.834 | 0.830 |
| collins | 22 | 0.960 | 1069.1 | **9** | 0.970 | 447.1 | 10.5 | 0.970 | **440.1** | 6 | 0.2 | 0.838 | 0.984 |
| fri_c4_500_100 | 100 | 0.784 | 5065.6 | **34** | **0.824** | 1757.0 | 35.3 | 0.790 | **1169.3** | 10 | 5.4 | 0.662 | 0.666 |
| gina_agnostic | 969 | 0.826 | 48137.2 | 87 | **0.889** | 4272.9 | **53** | 0.828 | **592.7** | 10 | 671.4 | 0.224 | 0.139 |
| glass | 9 | 0.645 | 476.4 | 0 | 0.645 | 0.0 | 0 | 0.645 | 0.0 | 10 | 0.0 | 1.000 | 1.000 |
| hillvalley | 100 | 1.000 | 4901.8 | **3** | 1.000 | 138.9 | 3.2 | 1.000 | **106.3** | 9 | 0.6 | 0.664 | 0.765 |
| ionosphere | 33 | 0.923 | 1696.4 | **10** | **0.934** | 533.5 | 11.7 | 0.928 | **455.0** | 9 | 0.2 | 0.778 | 0.853 |
| iris | 4 | 0.740 | 190.5 | **3** | 0.760 | **135.1** | 3.2 | **0.764** | 135.9 | 8 | 0.0 | 0.849 | 1.005 |
| jasmine1 | 144 | 0.761 | 7297.4 | 67 | **0.795** | 3495.3 | **9.9** | 0.767 | **179.7** | 10 | 6.2 | 0.363 | 0.051 |
| kc2 | 21 | 0.837 | 1126.0 | 3 | 0.837 | 153.9 | **1.3** | **0.842** | **45.2** | 10 | 0.0 | 0.696 | 0.294 |
| MegaWatt1 | 37 | 0.885 | 1802.1 | 0 | 0.893 | 0.0 | 0 | 0.893 | 0.0 | 10 | 0.0 | 1.000 | 1.000 |
| musk1 | 166 | 0.834 | 8189.6 | **19** | **0.882** | 972.5 | 21.5 | 0.843 | **559.4** | 10 | 1.8 | 0.520 | 0.575 |
| newthyroid | 5 | 0.921 | 240.5 | 5 | 0.926 | 240.5 | 5 | 0.926 | 240.5 | 10 | 0.0 | 0.962 | 1.000 |
| oil_spill | 28 | 0.956 | 1394.1 | 0 | 0.956 | 0.0 | 0 | 0.956 | 0.0 | 10 | 0.0 | 1.000 | 1.000 |
| pima | 8 | 0.688 | 408.0 | 6 | 0.691 | 297.9 | **4** | **0.711** | **186.4** | 10 | 0.1 | 0.932 | 0.626 |
| scene | 294 | 0.524 | 14660.5 | 30 | 0.748 | 1544.4 | **0.2** | **0.810** | **1.5** | 10 | 2.2 | 0.153 | 0.001 |
| segmentation | 19 | 0.997 | 990.1 | **3** | 0.998 | 149.2 | 3.1 | 0.998 | **131.3** | 8 | 0.4 | 0.847 | 0.880 |
| sonar | 60 | 0.841 | 3018.3 | 10 | **0.875** | 464.7 | **9** | 0.854 | **255.3** | 10 | 0.1 | 0.567 | 0.549 |
| SPECTF | 44 | 0.828 | 2253.2 | 11 | **0.868** | 572.4 | 11 | 0.841 | **336.5** | 10 | 0.2 | 0.612 | 0.588 |
| tokyo1 | 7 | 0.798 | 362.4 | 6 | **0.861** | 303.0 | **1.1** | 0.830 | **46.8** | 10 | 0.1 | 0.852 | 0.155 |
| vehicle | 18 | 0.942 | 905.8 | 11 | **0.965** | 560.4 | **8.9** | 0.961 | **341.5** | 10 | 0.3 | 0.767 | 0.609 |
| wdbc | 30 | 0.933 | 1498.1 | 4 | 0.938 | 228.3 | **2.3** | **0.946** | **102.4** | 10 | 0.1 | 0.891 | 0.449 |
| wine | 13 | 0.972 | 642.4 | **2** | 0.989 | **85.8** | 2.1 | 0.989 | 86.5 | 9 | 0.0 | 0.824 | 1.009 |
| wpbc | 30 | 0.856 | 1456.1 | 0 | 0.856 | 0.0 | 0 | 0.856 | 0.0 | 10 | 0.0 | 1.000 | 1.000 |
| yeast | 103 | 0.796 | 5059.1 | **32** | **0.814** | 1612.0 | 32.8 | 0.798 | **935.2** | 10 | 14.1 | 0.570 | 0.580 |
| *average:* | *83.7* | *0.8* | *4172.5* | *13.5* | ***0.874*** | *684.1* | ***9.1*** | *0.867* | ***259.2*** | *9.5* | *25.8* | *0.732* | *0.674* |

than for the min-cost incremental hyperplane, but the differences are small. The *Succ* (success) column gives the number of times out of the ten trials that the cost for the min-cost incremental hyperplane is less than or equal to the cost for the sparse hyperplane. On average, the min-cost incremental hyperplane has an equal or lower cost than the sparse hyperplane 9.5 times out of ten.

The final two columns (C*ost Ratio to Exp*, *Cost Ratio to Sparse*) give the ratio of the average cost for the min-cost incremental hyperplane to the expected cost and to the sparse hyperplane cost. The expected cost is calculated as the expected feature cost (50) times the average number of features in the min-cost hyperplane. The average ratio to the expected cost is 0.732 and to the sparse hyperplane cost 0.674. The method is effective in finding lower cost solutions where the features have costs.

Solution times are significantly smaller than for previous methods, especially for large datasets like *gina_agnostic*. This is mainly due to using a list length of 1.

# 4 Conclusions

Given an initial separation found by any hyperplane placement method, the *maxFS*-based methods developed here are extremely successful in finding an improved hyperplane that has equivalent or better accuracy while using significantly fewer features. The incremental version of the algorithm generally finds solutions using even fewer features in less time but does not guarantee to correctly classify the same set of points as correctly classified by the original hyperplane. The cost-minimizing version of the heuristic is also very effective.

The algorithms scale reasonably well. The longest run time for the incremental method is 4252.9 seconds for the massive *gina_agnostic* dataset (3,363,960 nonzeros). Future work will address speed ups for even larger datasets. There are several possibilities here:

- The experiments in this paper use a list length of 7 in most of the *maxFS*-based algorithms, which has worked well in previous experiments. It may be that using a list length of 1 is about as effective (and will take $1/7^{th}$ of the time).
- As Experiment 2 shows, limiting the number of features to half the number returned by the sparse solution does not have a big impact on the accuracy of the separation. Thus, imposing a time limit and using only the features selected by that time may not have a major impact on accuracy, even though fewer features are used. This may be helpful for massive datasets.
- Massive datasets can be sampled to reduce their size before applying the sparse hyperplane placement methods. The sampling can be made more useful by first running a $k$-nearest-neighbours calculation and including only data points that have in their neighbour set a point of the opposite type.

# 5 References

E Amaldi, M Pfetsch, L Trotter Jr. (1999). Structural and Algorithmic Properties of the Maximum Feasible Subsystem Problem, Proceedings of the Integer Programming and Combinatorial Optimization conference (IPCO'99), Lecture Notes in Computer Science 1610, Springer-Verlag, New York, NY, pp. 45–59.

PS Bradley, PL Mangasarian, WN Street (1998). Feature Selection via Mathematical Programming, INFORMS Journal on Computing 10:209–217.

JW Chinneck (1996). An Effective Polynomial-Time Heuristic for the Minimum-Cardinality IIS Set-Covering Problem, Annals of Mathematics and Artificial Intelligence 17:127–144.

JW Chinneck (2001). Fast Heuristics for the Maximum Feasible Subsystem Problem, INFORMS Journal on Computing, 13(3):210–223.

JW Chinneck (2007). Feasibility and Infeasibility in Optimization: Algorithms and Computational Methods (Vol. 118). Springer Science & Business Media.

JW Chinneck (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.

JW Chinneck (2012). Integrated classifier hyperplane placement and feature selection. Expert Systems with Applications, 39(9):8193–8203.

JW Chinneck (2018). Sparse Solutions of Linear Systems via Maximum Feasible Subsets, Les Cahiers du GERAD G–2018–104. https://www.gerad.ca/en/papers/G-2018-104.

JW Chinneck (2019). The Maximum Feasible Subset Problem (maxFS) and Applications, INFOR, to appear.

D Dua, and C Graff. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

S Jokar and ME Pfetsch (2008). Exact and Approximate Sparse Solutions of Underdetermined Linear Equations, SIAM Journal on Scientific Computing 31(1):23–44.

S Khalid, T Khalil, and S Nasreen (2014). A survey of feature selection and feature extraction techniques in machine learning. In 2014 Science and Information Conference (pp. 372–378). IEEE.

J Li, K Cheng, S Wang, F Morstatter, RP Trevino, J Tang, and H Liu (2018). Feature selection: A data perspective. ACM Computing Surveys (CSUR), 50(6):94.

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

Matlab (2019). https://www.mathworks.com/products/matlab.html.

Mosek (2019). https://www.mosek.com/.

OpenML (2019). OpenML Machine Learning Repository [https://www.openml.org/]. J. Vanschoren, J.N. van Rijn, B. Bischl, and L. Torgo. OpenML: networked science in machine learning. SIGKDD Explorations 15(2):49–60, 2013.

JK Sankaran (1993). A Note on Resolving Infeasibility in Linear Programs by Constraint Relaxation, Operations Research Letters 13:19–20.

J Tang, S Alelyani, and H Liu (2014). Feature selection for classification: A review. Data classification: algorithms and applications, 37.