MATHEMATICAL PROGRAMMING APPROACHES TO MACHINE LEARNING AND DATA MINING

By

Paul S. Bradley

A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

(COMPUTER SCIENCES)

at the

UNIVERSITY OF WISCONSIN – MADISON

1998

Abstract

Machine learning problems of supervised classification, unsupervised clustering and parsimonious approximation are formulated as mathematical programs. The feature selection problem arising in the supervised classification task is effectively addressed by calculating a separating plane by minimizing separation error and the number of problem features utilized. The support vector machine approach is formulated using various norms to measure the margin of separation. The clustering problem of assigning m points in n-dimensional real space to k clusters is formulated as minimizing a piecewise-linear concave function over a polyhedral set. This problem is also formulated in a novel fashion by minimizing the sum of squared distances of data points to nearest cluster planes characterizing the k clusters. The problem of obtaining a parsimonious solution to a linear system where the right hand side vector may be corrupted by noise is formulated as minimizing the system residual plus either the number of nonzero elements in the solution vector or the norm of the solution vector. The feature selection problem, the clustering problem and the parsimonious approximation problem can all be stated as the minimization of a concave function over a polyhedral region and are solved by a theoretically justifiable, fast and finite successive linearization algorithm. Numerical tests indicate the utility and efficiency of these formulations on real-world databases. In particular, the feature selection approach via concave minimization computes a separating-plane based classifier that improves upon the generalization ability of a separating plane computed without feature suppression. This approach produces classifiers utilizing fewer original problem features than the support vector machine approaches, with comparable generalization ability. The clustering techniques are shown to be effective and efficient data mining tools in medical survival analysis applications. The parsimonious approximation methods yield improved results in a signal processing application, with high signal to noise ratio, over least squares and a lengthy combinatorial search. These results support the claim that mathematical programming is effective as the basis of data mining tools to extract patterns from a database which contain "knowledge" and thus achieve "knowledge discovery in databases".

Acknowledgements

My greatest thanks go to my wife, Kate. Her constantly growing love and support has brightened every day. I am extremely grateful to my parents, Mary and Stephen Bradley. Their example, support, encouragement and love have provided the basis for the place I stand today. I also thank my sister Sarah who has always gifted me with a smile.

My utmost gratitude and respect go to Olvi Mangasarian. It has been a true honor to work with an excellent mentor, brilliant mathematician and friend. I thank Grace Wahba for openly accepting me into her weekly graduate student mini-seminars and sharing her knowledge regarding smoothing splines and regularization. I thank Michael Ferris and Robert Meyer for sharing their mathematical optimization expertise. I gratefully thank Jude Shavlik for providing a basic understanding of machine learning and related methodology. I also thank these people for serving on my thesis committee, their insights greatly improved this work.

I also wish to thank Nick Street, whom I have followed.

This research is supported by National Science Foundation Grants CCR-9322479, CCR-9729842 and CDA-9623632, and by Air Force Office of Scientific Research Grant F49620-97-1-0326.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

In this work, we explore mathematical programming approaches to supervised and unsupervised machine learning problems. We further integrate these approaches into the broader goal of extracting knowledge from databases.

1.1 Machine Learning: Supervised Learning

In a *supervised learning* task, one attempts to estimate a function g which maps points or feature vectors from an *input space* $\mathcal{X}$ to an *output space* $\mathcal{Y}$, given only a finite sample of the mapping $\{x^i, g(x^i)\}_{i=1}^{M}$, which may be corrupted with noise. The goal of a supervised learning algorithm is to construct an estimate $\hat{g}$ of g from this finite sample, or *training set*.

Our principal supervised learning task is the *classification problem*. The classification problem is usually defined as that of assigning a vector $x \in \mathcal{X}$ to one of k disjoint subsets $\mathcal{A}_1$,$\mathcal{A}_2, \ldots$,$\mathcal{A}_k$ of $\mathcal{X}$. We address the classification task in its simplest form: determine whether an input vector $x \in \mathcal{X}$ is an element of one of the disjoint point sets $\mathcal{A}$ or $\mathcal{B}$, where $\mathcal{A}$,$\mathcal{B} \subset \mathcal{X}$. We assume the true classification function g has the following form:

$$g(x) = \begin{cases} 1 & \text{if } x \in \mathcal{A} \\ \\ 0 & \text{if } x \in \mathcal{B}. \end{cases} \tag{1}$$

Many different algorithms exist for constructing the approximation $\hat{g}$ of g and the approximation may have many different functional forms. Examples include a separating-plane based function [96, 97], the backpropagation algorithm for artificial neural networks (ANNs) [77, 111], decision tree construction algorithms utilizing various node-decision criteria [29, 134, 5], spline methods for classification [165, 162] and probabilistic graphical dependency models [76, 32].

Evaluating an estimate $\hat{g}$ of g in terms of how well it performs on data not included in the training set, or how well $\hat{g}$ *generalizes*, is paramount. Often it is possible to allow a learning algorithm to construct $\hat{g}$ from a sufficiently complex function class so that $\hat{g}$ approximates g arbitrarily well on the training set. However this complex $\hat{g}$ usually approximates g poorly on points not in the training set [148]. This situation is identified as *overfitting* the training data. While *biasing* a learning algorithm to construct $\hat{g}$ from a less complex function class often improves generalization ability, it may not be desirable in all problem domains [142]. *Overtraining* can also lead to poor generalization even when the complexity of the function class from which $\hat{g}$ is constructed is optimal [15]. The key to good generalization is correctly estimating the complexity of the true mapping g while avoiding overtraining. This problem is compounded by the fact that we have only a finite sampling of g, which, in addition, may be corrupted by noise.

To avoid overfitting, many learning algorithms utilize the *Occam's Razor* bias [13] (or equivalently, *minimum description length* [136] or *minimum message length* [166]). This bias favors the simplest model possible that still achieves some level of performance on the training data. Common techniques to avoid overtraining are early stopping and weight decay [16]. The latter can also be viewed as an overfitting avoidance technique.

Generalization ability can be effectively estimated by *leave-one-out testing* [90], where $\hat{g}$ is constructed by a learning algorithm using all but one of the available training examples and is tested on the point "left out". This procedure is repeated for each data point in the training set. The average correctness measure over all of the data points tested is an accurate, nearly unbiased estimate of the generalization ability of $\hat{g}$. A method which is computationally more tractable for estimating generalization ability is *cross-validation* [152]. In this procedure, the available training data is randomly divided into ℓ disjoint sets of approximately equal size, $T_1, T_2, \ldots, T_\ell$. Then ℓ trials are conducted, where in each trial, the test set is T_i, and the training set is the union of all T_j, $j \neq i$, $i = 1, \ldots, \ell$. A further discussion of statistical tests for comparing different learning algorithms which construct estimators $\hat{g}$ can be found in [53].

1.2 Machine Learning: Unsupervised Learning

The fundamental difference between an unsupervised learning task and a supervised learning task is that the elements from the input space $\mathcal{X}$ are *not* tagged with their corresponding element in the output space $\mathcal{Y}$ in unsupervised learning. The motivation for an unsupervised learning algorithm is to "look for regularities" in the training examples $\{x^i\}_{i=1}^M \subset \mathcal{X}$ [148]. Unsupervised learning algorithms can be divided into general "discovery" systems (e.g. AM [93], BACON [91]) and those which perform "clustering" [148].

In the clustering problem [81, 63], the goal is to group or cluster the data into sets of "like" points. One hopes to obtain clusters revealing some sort of high-level characterization of the points belonging to individual clusters. "Exemplar" or "prototype"-based clustering approaches include Forgy's method [61], the MacQueen algorithm [95] (commonly referred to as "batch" and "online" k-Mean clustering), Kohonen maps [87], and competitive learning [141]. Probabilistic clustering methods include the COBWEB algorithm [60], AutoClass [39], the Expectation-Maximization (EM) algorithm [50, 124] and, more recently, probabilistic graphical approaches [76, 32].

"Hard" clustering algorithms, such as k-Mean, assign data items to a single cluster whereas *"soft" clustering* algorithms, such as EM, assign a given data point to all clusters with a certain probability of membership. Hard clustering algorithms can often be placed in the probabilistic framework of soft clustering [18, 17].

Formal methodology for evaluating clustering algorithms is lacking, in contrast to the fairly standard methodologies of leave-one-out testing and ten-fold cross-validation used in the supervised classification problem. Evaluation procedures include distortion (if applicable), information gain [18], classification accuracy (if classes are known) and holdout likelihood [112].

Generalization ability is usually not an explicit issue in unsupervised learning. Although generalization ability is not paramount, clustering methods still implement a bias when defining clusters. This bias may involve a restricted cluster description or place an ordering over various possible cluster descriptions [148].

In the next section we introduce a process for extracting useful information from collected data utilizing supervised and unsupervised machine learning methods.

1.3 Knowledge Discovery in Databases

Knowledge discovery in databases (KDD) has been defined as "... the nontrivial process of identifying valid, novel, potentially useful and ultimately understandable patterns in data" [57]. This multi-disciplinary field is driven by the need to extract useful information from enormous amounts of data collected in areas ranging from astronomy [56] to business and industrial domains [130].

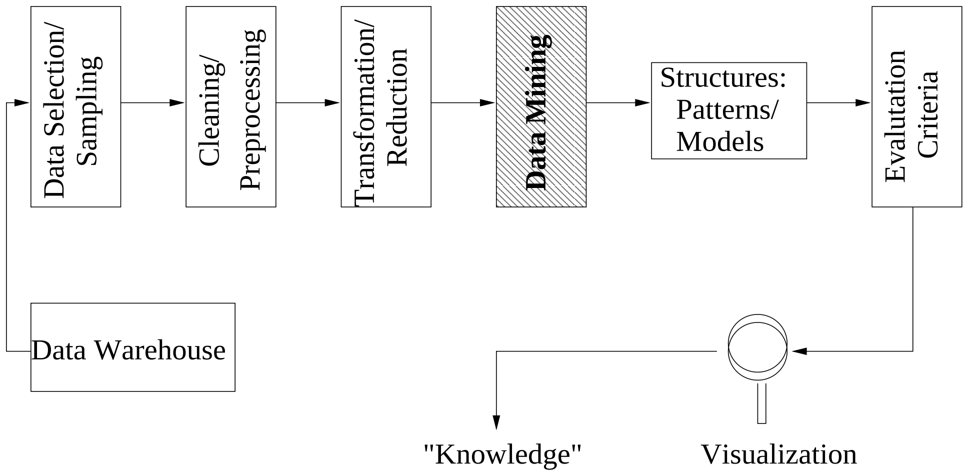


Figure 1: An overview of the steps comprising the KDD process.

The *KDD process* consists of numerous interactive and iterative steps enabling "knowledge" to evolve from a given store of "raw" data. These steps include data selection, preprocessing and transformation to put the raw data in a form suitable for the application of one (or more) *data mining* algorithms. Any algorithm that enumerates patterns from, or fits models, to data is a *data mining algorithm* [58]. Many algorithms from pattern recognition, statistics, databases, machine learning and optimization qualify as data mining algorithms and give the field of KDD its multi-disciplinary character. Related fields further include high performance and parallel computing, knowledge modeling, management of uncertainty, and data visualization [58].

After the data mining algorithm has identified patterns (or computed models) of possible interest, this output must be interpreted or evaluated. Evaluation criteria include *certainty* (estimated predictive accuracy on unseen data or generalization ability) or *utility* (gain, perhaps in dollars saved due to better predictions or speed-up in response time of a system). Other criteria such as *novelty* and *understandability* are much more subjective and difficult to define [21]. Application of a visualization tool may aid in evaluating the output of the data mining algorithm. See Figure 1.

Although not explicitly detailed in Figure 1, the KDD process is typically iterative and previous steps may need to be altered and algorithms re-applied to obtain patterns or models deemed useful. For example, one may select, clean and reduce data only to discover after mining that one or several of the previous steps need to be redone [21].

Depending on the goals of the user, different data mining methods may be employed in the KDD process. Two primary data mining methods are classification and clustering [57].

1.3.1 Massive Dataset Issues

Many popular classification and clustering algorithms [148, 63, 11, 81] require accessing individual data items an arbitrary number of times, or require data to be resident in memory for effective processing. These approaches play powerful roles as data mining algorithms over data sets of modest size.

The issue of scalability of these algorithms arises when datasets become massive (too large for resident memory) and/or are stored remotely (access becomes expensive). In fact, massive dataset size is a fundamental characteristic of many working definitions of KDD [151]. Data can grow in the number of data elements (records) and in the dimensionality of each element (fields per record).

There are straightforward ways to "scale" current algorithms to deal with large scale or massive datasets. A standard approach to dealing with high-dimensional data is to project it into a low-dimensional space and attempt to construct models in this space [21]. Three drawbacks to this scheme are: (1) as the number of dimensions grow, the combinatorial choice of projection into a space of lower dimension becomes overwhelming; (2) projection to a space of lower dimension could transform a relatively easy modeling problem into one that is extremely difficult [21]; (3) the projection may result in loss of accuracy or "quality", resulting in a poor model or a difficult problem (as in (2)). For datasets with a massive number of examples, the straightforward way to "scale" existing algorithms is by random sampling. One selects a random sample that fits into resident memory and constructs models over this sample. Models constructed over random samples may be inadequate [19, 18]. These drawbacks may be overcome by employing more complex sampling or "averaging" schemes.

To deal efficiently with massive datasets, we propose that data mining algorithms be developed which satisfy (to the extent possible) the following [19]:

- *Require one scan of the dataset if possible:* since even a single data scan over a massive database can be costly. Early termination, if appropriate, is highly desirable.

- *On-line "anytime" behavior [14]:* a "best" solution is always available, with status information on progress provided to allow the user to terminate the process if the current solution is adequate.

- *Suspendable, stoppable, resumable:* incremental progress is saved to resume a stopped job.

- *Ability to incrementally incorporate additional data* with existing models efficiently. In many data mining applications, data is provided in a temporal manner (e.g. the database is updated daily).

- *Work within the confines of a limited resident memory (RAM) buffer.*

In the next section we introduce the notation that will be used throughout this work.

1.4 Notation

- All vectors will be column vectors unless transposed to a row vector by the use of a prime superscript $'$.

- The symbol ":=" denotes definition.

- For a vector x in the n-dimensional real space R^n, x_+ will denote the vector in R^n with components $(x_+)_i := \max\{x_i, 0\}$, $i = 1, \ldots, n$.

- The vector x_* will denote the vector in R^n with components $(x_*)_i := (x_i)_*$, $i = 1, \ldots, n$, where $(\cdot)_*$ is the step function defined as one for positive x_i and zero otherwise.

- The notation $|x|$ will denote a vector of absolute values of components of x.

- The base of the natural logarithm will be denoted by ε and for $y \in R^m$, ε^{-y} will denote a vector in R^m with components ε^{-y_i}, $i = 1, \ldots, m$.

- The norm $\|\cdot\|_p$ will denote the p norm, $1 \leq p \leq \infty$. For a general norm $\|\cdot\|$ on R^n, the dual norm $\|\cdot\|'$ on R^n is defined as

$$\|x\|' = \max_{\|y\|=1} x'y.$$

  The 1-norm and ∞-norm are dual norms, and so are a p-norm and q-norm for which $1 \leq p,\ q \leq \infty$ and $\frac{1}{p} + \frac{1}{q} = 1$.

- The notation $A \in R^{m \times n}$ will signify a real $m \times n$ matrix. For such a matrix, A' will denote the transpose, A_i will denote row i and $A_{\cdot j}$ will denote column j.

- For two vectors x and y in R^n, $x \perp y$ will denote $x'y = 0$.

- A vector of ones in a real space of arbitrary dimension will be denoted by e.

- The notation $\arg\min\limits_{x \in S} f(x)$ will denote the set of minimizers of $f(x)$ on the set S. Similarly $\arg \mathrm{vertex} \min\limits_{x \in S} f(x)$ will denote the set of vertex minimizers of $f(x)$ on the polyhedral set S, that is the set of vertices of S that solve $\min\limits_{x \in S} f(x)$.

- By a separating plane, with respect to two given point sets $\mathcal{A}$ and $\mathcal{B}$ in R^n, we shall mean a plane that attempts to separate R^n into two half spaces such that each open halfspace contains points mostly of $\mathcal{A}$ or $\mathcal{B}$. Alternatively, such a plane can also be interpreted as a classical perceptron [140, 77, 100].

- For a function $f : R^n \to R$ that is concave on R^n, the supergradient $\partial f(x)$ of f at x is a vector in R^n satisfying

$$f(y) - f(x) \leq \partial f(x)(y - x) \tag{2}$$

for any $y \in R^n$. The set $D(f(x))$ of supergradients of f at the point x is nonempty, convex, compact and reduces to the ordinary gradient $\nabla f(x)$, when f is differentiable at x [132, 137].

Chapter 2

Classification via Mathematical Programming

We focus on optimization approaches addressing the supervised classification task (Section 1.1). The task is that of assigning a point $x = [x_1\, x_2\, \ldots\, x_n]'$ in n-dimensional feature space into one of two disjoint point sets $\mathcal{A}$ or $\mathcal{B}$ by estimating a classification function $\hat{g}$ from a finite training set $\{x^i, g(x^i)\}_{i=1}^{M}$. Here, $g(x)$ is either 1 or 0, depending on whether $x \in \mathcal{A}$ or $x \in \mathcal{B}$, respectively. Difficulty in estimating g may stem from the fact that the training set may have errors or is not entirely representative of data points to be encountered in the future.

We begin with the feature selection problem which consists of eliminating as many features from the original n-dimensional features space as possible, while still accurately estimating g over the training set. We then focus on the support vector machine [22, 33, 161] approach to classification. Computational comparison is made between a feature selection approach via concave minimization and the support vector machine formulations. A technique for solving general linear programs with a massive number of constraints is considered. This technique, when applied to the linear formulations of the feature selection and support vector machine problems, addresses the issues of massive datasets often encountered in many large scale KDD applications (Section

1.3.1).

2.1 Feature Selection Problem

The accuracy of the classification function estimated from the training data, $\hat{g}$, is determined by the inherent class information available in the features, or components, of the input vectors x^i in the training set. It seems logical to conclude that having a large number of features would provide more discrimination ability. But, with a large number of features and a fixed number of training examples, one encounters the "curse of dimensionality"; a high-dimensional space with a modest number of data points is almost empty [77]. Thus, with a large number of original problem features and a finite amount of training data, *many* different estimators may accurately separate the training data, but only a small fraction may generalize well.

In the classification task described above, one realization of the Occam's Razor bias [13] is to "choose" a small number of predictive features and utilize these to construct the estimator $\hat{g}$, discarding irrelevant or redundant features. Removal of irrelevant or redundant features usually speeds the learning process, the constructed estimator usually generalizes better and often lends itself to easier interpretation [117]. In addition, a classification function utilizing a small number of original problem features is often requires fewer resources to evaluate.

The feature selection problem consists of eliminating as many features from the original n-dimensional feature space as possible, while still accurately estimating g to some extent over the training set.

2.1.1 Machine Learning Approaches

Machine learning approaches to feature subset selection can basically be divided along the lines of *filter models* and *wrapper models* [83]. In the filter model, feature selection is done as a preprocessing step which ignores the effects of the selected subset on the performance of the estimator constructed by a given algorithm. In contrast, the wrapper model encompasses algorithms which utilize the estimator $\hat{g}$ to evaluate a given feature subset.

Instances of the filter model include the FOCUS algorithm [4], the Relief algorithm [85] and a method of feature selection based upon Information Theory [88].

The FOCUS algorithm [4] performs an exhaustive search through the possible feature subsets and returns the minimal subset sufficient for classification. FOCUS may not be able to identify irrelevant features when noise is present in the training set, or if the training set is not representative of future data [83].

The Relief algorithm [85] assigns a weight to each feature which corresponds to the "relevance" of that feature as represented in the training set. Examples from the training set are randomly selected and the relevance value is calculated based upon the component-wise distance from the selected point to the two nearest neighbors of the same and opposite class. The relevance of a feature is then an average of these values over the sampled points from the training set. Since each feature relevance is determined independently of other features, Relief is unable to identify redundant features [85]. In a domain in which a relevant group of features are highly correlated, Relief will assign to each of the correlated features a large relevance value and not notice the redundancy.

The feature selection method proposed in [88] attempts to compute a reduced feature set such that the probability distribution of the class given the reduced feature set is "near" to the distribution of the class given the full feature set. Consider a particular data instance characterized by given values of the original full set of problem features. This data instance induces a probability distribution of the class label given the values of all of the problem features. The idea is to remove a feature such that the probability distribution of the class given the reduced feature subset is as "near" to the distribution of the class given the full feature subset. Distance here is the KL-distance [89] or equivalently, cross-entropy.

A few examples of the wrapper model are forward selection (backward elimination), forward stepwise selection (backward stepwise elimination), Optimal Brain Damage and SET-GEN.

One of the simplest hillclimbing feature selection methods is *forward selection* [36]. Initially, the candidate set of features is empty. At each iteration, one feature is greedily added to the candidate set as the one that provides the maximum increase in a generalization estimate . *Backward elimination* [36] begins with the candidate set of features consisting of all original problem features. At each iteration, one feature is greedily removed from the candidate set so that the resulting smaller subset corresponds to a maximum increase in estimated generalization. Iterations cease when a given generalization increase is not observed. Notice that in forward selection, once a feature has been added to the candidate set, it will not be removed. Similarly, in backward elimination, once a feature has been removed, it cannot again re-enter the candidate set. Note that the method of [88] is a backward-elimination scheme which scores a given feature subset via cross-entropy or KL-distance [89] instead of utilizing the performance of the classifier computed by the induction algorithm.

Forward stepwise selection [36] is a form of bidirectional hillclimbing in that at each step, one feature is allowed to be added and one feature is removed from the candidate subset. Forward stepwise selection differs from *backward stepwise elimination* [36] only in that the former begins with an empty candidate set and the latter with a candidate set consisting of all original features. These searches must be monitored to prevent cycling. There are further generalizations of these basic procedures such as backward stepwise elimination–SLASH (BSE–SLASH) [36] and Greedy Sequential Backward Elimination (GBSE) [30].

Optimal Brain Damage (OBD) [92] is a procedure to set the weights of an artificial neural network (ANN) to zero, but when applied to weights associated with input features, this is synonymous with feature selection. The idea of OBD is to delete parameters with small "saliency", or parameters whose deletion will have minimal effect on training error. The error function is assumed to be twice differentiable. The diagonal terms of the Hessian of this error function (times a multiplier) which are less than a given tolerance correspond to parameters with small saliency. The OBD procedure can be summarized as follows: (1) A classifier is trained until a "reasonable" solution is obtained, (2) second derivatives of the error function with respect to each parameter (diagonal elements of the Hessian) are computed, (3) saliencies are determined for each of these parameters, and (4) the parameters with saliencies below a given threshold are set to zero. With the given parameters set to zero, steps (1)-(4) are repeated.

SET-GEN [42] employs a genetic algorithm [71] to search candidate feature subsets. For each candidate subset, a C4.5 decision tree [134] is constructed using the given candidate input features and generalization ability of the decision tree is estimated by ten-fold cross-validation [152]. The fitness of each candidate is then determined by a linear combination of the generalization estimate, number of candidate features, and average size of trees over each fold of cross-validation. A population is maintained of the best candidate subsets. Genetic operators are used to create new subsets. If a new candidate subset has better fitness than the worst one in the current population, it is replaced. After a desired number of subset evaluations are completed, SET-GEN uses the entire training set to produce a C4.5 tree using the best candidate subset.

Some inductive algorithms, such as those producing decision trees with univariate splits (e.g. ID3 [133], C4.5 [134]), inherently select features while constructing $\hat{g}$. For instance, if the criterion for adding a new node is satisfied with ID3, the feature on which to base this decision is chosen as one that maximizes a measure of information gain. While known for being a very fast procedure, ID3 suffers when the true relation between input/output pairs utilizes interactions between features [51, 129] since it does not make use of more than one feature in any given decision.

2.1.2 Mathematical Programming Approaches

As mentioned in Section 2.1, our task is to discriminate between two given sets in an n-dimensional feature space using as few of the given features as possible.

Formulating the feature selection problem as a mathematical program has been extensively studied. In [118], stepwise techniques are utilized to avoid exhaustive enumeration of all possible feature subsets. In [38], dynamic programming is used for the same purpose. In [123], a branch and bound method is presented that effectively rejects suboptimal subsets without direct evaluation and yields a global solution. In [28], the feature selection problem is formulated as a linear program with equilibrium (complementarity) constraints (LPEC), but both the LPEC formulation and its method of solution differ from those presented here.

In this section, we consider mathematical programming approaches for estimating the classification function given a training set consisting of two nonempty, disjoint, finite point sets $\mathcal{A}$ and $\mathcal{B}$ in n-dimensional feature space with m and k points respectively. The point sets $\mathcal{A}$ and $\mathcal{B}$ are represented by the matrices $A \in R^{m\times n}$ and $B \in R^{k\times n}$, where each point of $\mathcal{A}$ is represented by a row in A, similarly for $\mathcal{B}$. We attempt to discriminate between the points of $\mathcal{A}$ and $\mathcal{B}$ by constructing a separating plane:

$$P := \{x \,|\, x \in R^n, x'w = \gamma\}, \tag{3}$$

with normal $w \in R^n$ and (2-norm) distance $\frac{|\gamma|}{\|w\|_2}$ to the origin, while suppressing as many elements of the vector w as possible. We want to determine w and γ so that the separating plane P determines two open halfspaces: $\{x \,|\, x \in R^n, x'w > \gamma\}$ containing mostly points of $\mathcal{A}$, and $\{x \,|\, x \in R^n, x'w < \gamma\}$ containing mostly points of $\mathcal{B}$. Hence we wish to satisfy

$$Aw > e\gamma,\ Bw < e\gamma \tag{4}$$

to the extent possible. Upon normalization, these inequalities can be equivalently written as

$$Aw \geq e\gamma + e,\ Bw \leq e\gamma - e. \tag{5}$$

Conditions (4), or equivalently (5), can be satisfied if and only if, the convex hulls of $\mathcal{A}$ and $\mathcal{B}$ are disjoint. This is not the case in many real-world applications. Hence, we attempt to satisfy (5) in some "best" sense, for example, by minimizing some norm of the average violations of (5) such as

$$\min_{w,\gamma} \; f(w,\gamma) := \min_{w,\gamma} \; \frac{1}{m}\|(-Aw + e\gamma + e)_+\|_1 + \frac{1}{k}\,\|(Bw - e\gamma + e)_+\|_1 \qquad (6)$$

Another way of satisfying (5) consists of minimizing the number of points misclassified by the separating plane P [101, 121]. In [40], the decision problem associated with minimizing the number of misclassifications is shown to be NP-complete and a hybrid algorithm is proposed to compute an approximate solution.

Two principal reasons for choosing the 1-norm in (6) are:

(i) Problem (6) is then reducible to a linear program (7) with many important theoretical properties making it an effective computational tool [8].

(ii) The 1-norm is less sensitive to outliers such as those occurring when the underlying data distributions have pronounced tails, hence (6) has a similar effect to that of robust regression [78],[75, pp 82-87].

The formulation (6) is equivalent to the following robust linear programming formulation (RLP) proposed in [7] and effectively used to solve problems from real-world domains [109]:

$$\min_{w,\gamma,y,z} \left\{ \frac{e'y}{m} + \frac{e'z}{k} \,\middle|\, -Aw + e\gamma + e \leq y,\; Bw - e\gamma + e \leq z,\; y \geq 0,\; z \geq 0 \right\}. \qquad (7)$$

Non-symmetric misclassification costs can easily be handled by the RLP formulation (7) by introducing parameters β_1 and β_2 into the objective. For instance, if the cost of misclassifying a point of $\mathcal{A}$ is greater than misclassifying a point of $\mathcal{B}$, this is easily reflected by replacing the objective of (7) with $\frac{(\beta_1)e'y}{m} + \frac{(\beta_2)e'z}{k}$, $\beta_1 > \beta_2$. This situation may arise in a classification problem were a "false-negative" response is deemed more costly than a "false-positive" response.

The linear program (7), or equivalently, the formulation (6) define a separating plane P that approximately satisfies the conditions (5).

To address the feature selection problem, we wish to find a minimum support solution [106] with respect to w of the RLP (7). Minimum support solutions are those with as many components equal to zero as possible. In order to suppress as many elements of w as possible, we introduce a feature selection term with parameter $\lambda \in [0, 1)$ into the objective of (7) and weight the original objective function by $(1 - \lambda)$ as follows:

$$\min_{w,\gamma,y,z} \left\{ (1-\lambda)(\frac{e'y}{m} + \frac{e'z}{k}) + \lambda e' \mid w \mid_* \;\middle|\; \begin{array}{c} -Aw + e\gamma + e \leq y, \\ Bw - e\gamma + e \leq z, \\ y \geq 0,\ z \geq 0 \end{array} \right\}, \lambda \in [0, 1). \tag{8}$$

The value of the parameter λ balances the two objectives of minimizing the error in separating the training data versus the number of original problem features used.

The feature suppression term $e'|w|_*$ counts the number of nonzero elements of the weight vector w defining the separating plane P.

The absolute value function is removed from the objective of (8) by introducing the variable $v \in R^n$ and adding the constraints $-v \leq w \leq v$ to the constraints of (8):

$$
\text{(FS)} \min_{w,\gamma,y,z,v} \left\{ (1-\lambda)(\frac{e'y}{m} + \frac{e'z}{k}) + \lambda e'v_* \left| \begin{array}{c} -Aw + e\gamma + e \leq y, \\ Bw - e\gamma + e \leq z, \\ y \geq 0, z \geq 0, \\ -v \leq w \leq v \end{array} \right. \right\}, \ \lambda \in [0,1). \qquad (9)
$$

This is our fundamental feature selection (FS) problem. At a solution of (9), the vector v_* is equal to $|w|_*$. Also note that the constraints $-v \leq w \leq v$ imply $v \geq 0$. The value of the parameter $\lambda \in [0,1)$ is chosen to maximize the generalization ability of the estimated classification function, $\hat{g}(x) := (x'w - \gamma)_*$, where w, γ are solutions of (9). Typically, this will be achieved in a feature space of reduced dimensionality, that is when the number of nonzero elements of the weight vector $w = e'v_* < n$.

Since the function $e'v_*$ is discontinuous on the non-negative orthant, we make a continuous approximation of it using either the standard sigmoid function of neural networks [139, 77] or by a concave exponential on the nonnegative real line [103]. The two approximations of the step vector v_* of (9) by the sigmoid function and the concave exponential are respectively:

$$
v_* \cong s(v,\alpha) := \left(e + \varepsilon^{-\alpha v}\right)^{-1}, \ \alpha > 0 \qquad (10)
$$

See Figure 2.

$$
v_* \cong t(v,\alpha) := e - \varepsilon^{-\alpha v}, \ \alpha > 0 \qquad (11)
$$

See Figure 3.

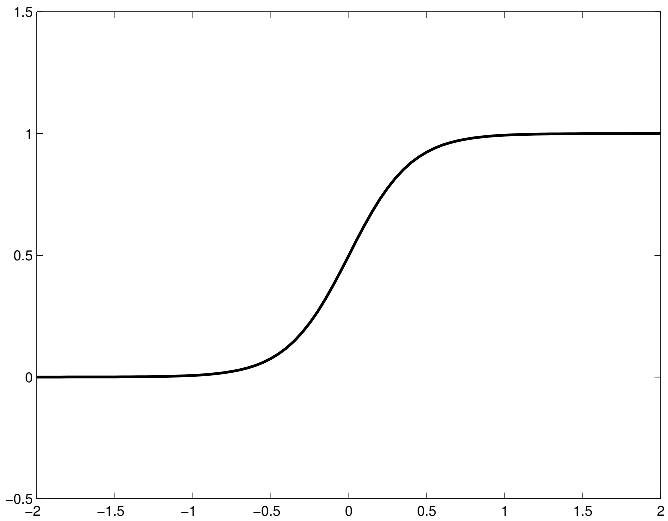


Figure 2: Sigmoid approximation (10) of the step function, $(\cdot)_*$. Plot is $s(x, 5) = \dfrac{1}{1 + \varepsilon^{-5x}}$ versus x.

Here e is a vector of ones, ε is the base of the natural logarithm, α is a positive parameter, and the application of either function to a vector is interpreted component-wise as in the standard MATLAB notation [110]. Note that for $\varepsilon^{-\alpha v} << e$, $e - \varepsilon^{-\alpha v}$ is an approximation to $(e + \varepsilon^{-\alpha v})^{-1}$. To see this, $1 - \delta$ is a two-term Taylor series expansion of $(1 + \delta)^{-1}$ for a scalar $\delta << 1$.

Advantages of the concave exponential (11) over the standard sigmoid (10) are the following:

(i) The concave exponential more accurately approximates the step function v_* at 0, since $t(0, \alpha) = 0$, whereas $s(0, \alpha) = \dfrac{1}{2}e$.

(ii) The concavity of $t(v, \alpha)$ in v on R^n leads to a finitely terminating algorithm (Algorithm 2.1.3).

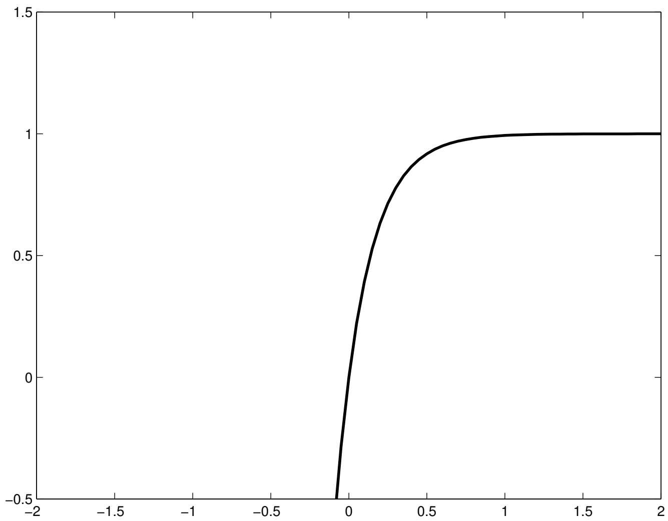


Figure 3: Concave exponential approximation (11) of the step function, $(\cdot)_*$. Plot is $t(x,5) = 1 - \varepsilon^{-5x}$ versus x. Recall that we are interested in approximating the step function on the non-negative real line.

With these approximations, the FS problem (9) can be approximated by the following FSS (FS sigmoid) and FSV (FS concave) problems:

$$
\text{(FSS)} \min_{w,\gamma,y,z,v} \left\{ (1-\lambda)\left(\frac{e'y}{m} + \frac{e'z}{k}\right) + \lambda e'(e + \varepsilon^{-\alpha v})^{-1} \;\middle|\; \begin{array}{c} -Aw + e\gamma + e \le y, \\ Bw - e\gamma + e \le z, \\ y \ge 0, z \ge 0, \\ -v \le w \le v \end{array} \right\},
$$

$$\lambda \in [0,1), \quad (12)$$

$$\text{(FSV)} \min_{w,\gamma,y,z,v} \left\{ (1-\lambda)(\frac{e'y}{m} + \frac{e'z}{k}) + \lambda(n - e'\varepsilon^{-\alpha v}) \;\middle|\; \begin{array}{c} -Aw + e\gamma + e \leq y, \\ Bw - e\gamma + e \leq z, \\ y \geq 0, z \geq 0, \\ -v \leq w \leq v \end{array} \right\},$$

$$\lambda \in [0,1). \quad (13)$$

We make use of the following lemma [103, Lemma 2.3] in order to model the step function of (9) exactly, as a linear program with equilibrium constraints (LPEC) [101, 94]. (The "equilibrium" terminology refers to the term involving the $\perp$ condition which characterizes complementarity problems [47].)

Lemma 2.1.1 *Let $a \in R^m$. Then*

$$r = a_*,\ u = a_+ \Leftrightarrow (r,u) = \arg\min_{r,u} \{e'r \mid 0 \leq r \perp u - a \geq 0,\ 0 \leq u \perp -r + e \geq 0\} \quad (14)$$

Utilizing this lemma, the FS problem can be equivalently rewritten as the following LPEC:

$$
\text{(FSL)} \min_{w,\gamma,y,z,v} \left\{ (1-\lambda)(\frac{e'y}{m} + \frac{e'z}{k}) + \lambda e'r \;\middle|\; \begin{array}{c} -Aw + e\gamma + e \leq y, \\ Bw - e\gamma + e \leq z, \\ y \geq 0, z \geq 0, \\ -v \leq w \leq v, \\ 0 \leq r \perp u - v \geq 0 \\ 0 \leq u \perp -r + e \geq 0 \end{array} \right\},
$$

$$
\lambda \in [0,1). \quad (15)
$$

One consequence of this LPEC formulation of the FS problem (9) is the existence of a solution to the FS problem [103, Proposition 2.5]. However, the FSL problem (15) is computationally difficult, in fact LPECs in general are NP-hard since they subsume the general linear complementarity problem which is NP-complete [43]. To avoid this difficulty, (15) is reformulated as a parametric bilinear program which can be easily handled by solving a finite sequence of linear programs that terminate at a stationary point [9, Algorithm 2.1]. In particular, the nonnegative nonlinear terms $r'(u-v)$ plus $u'(-r+e)$ are moved as positive penalty terms into the objective function as $-r'v+e'u$, weighted by the penalty parameter $\mu \in (0,1)$ as follows:

$$(\text{FSB}) \quad \min_{w,\gamma,y,z,v,u,r} \left\{ \begin{array}{c|c} & -Aw + e\gamma + e \le y, \\ & Bw - e\gamma + e \le z, \\ (1-\mu)((1-\lambda)(\frac{e'y}{m} + \frac{e'z}{k}) + \lambda e'r) & y \ge 0, z \ge 0, \\ +\mu(-r'v + e'u) & -v \le w \le v, \\ & 0 \le r, u - v \ge 0 \\ & 0 \le u, -r + e \ge 0 \end{array} \right\},$$

$$\lambda \in [0,1), \mu \in (0,1). \quad (16)$$

To summarize, we have formulated the feature selection problem as a mathematical program (9), which consists of finding a minimum support solution to the robust linear program (7). This is accomplished by suppressing the components of w by use of the step function $(\cdot)_*$. The step function is modeled exactly when the term $-r'v + e'u = 0$ at a solution to the FSB problem (16). It is approximated by the sigmoid function (10) in the FSS problem (12). The FSS problem consists of minimizing a nonlinear objective function over a polyhedral region. The step function is also approximated by the negative exponential (11) in the FSV problem (13). The FSV problem consists of minimizing a concave objective over a polyhedral region.

Algorithms to solve these feature selection formulations are discussed next.

2.1.3 Algorithms for the Feature Selection Problem

By using the standard transformation $w = w^1 - e\zeta^1$, $\gamma = \gamma^1 - \zeta^1$, we replace the variables (w, γ) by the nonnegative variables (w^1, γ^1, ζ^1). The feature selection problems FSS (12), FSV (13) and FSB (16) can then be transformed into the following minimization problem:

$$\min_{x}\{f(x) \mid Ax \leq b, x \geq 0\} \tag{17}$$

where $f : R^{\ell} \rightarrow R$, is a differentiable, nonconvex function bounded below on the nonempty polyhedral feasible region of (17), with $A \in R^{p\times\ell}$ and $b \in R^{p}$. Although this transformation is needed in order to establish theoretically the finiteness of our algorithms (the SLA 2.1.3 and the Bilinear Algorithm 2.1.5), we shall not carry it out in the interest of keeping the algorithms simple. In practice, finite termination occurs without this transformation.

We note that for each of the FSS (12), FSV (13) and FSB (16) problems, once the algorithms for computing solutions have terminated, the RLP (7) is re-solved with all non-selected features removed.

We consider the algorithms which compute solutions to the FSS, FSV and FSB problems to be related to wrapper models of feature selection [83]. As the FSS, FSV and FSB problems are being solved, the benefit of removing a problem feature (i.e. setting a given w_j to zero) is balanced by an increase (if any) in the term measuring separation error of the separating plane $w'x = \gamma$. In this sense the separation ability of the resulting classifier is a factor in selecting a given feature subset.

Because the FSS problem (12) objective is neither convex nor concave, it was solved using the nonlinear optimization codes available in the MINOS optimization package [120].

Polyhedral Concave Minimization

We consider a finitely terminating fast successive linearization algorithm for computing minimum support solutions to polyhedral concave programs. This approach finds minimum support solutions [106] to the following problem:

$$\min_{x \in S} f(x) \tag{18}$$

where $f : R^\ell \to R$ is a concave function on R^ℓ that is bounded below on the polyhedral set S not containing straight lines going to infinity in both directions. A minimum support solution to (18) is obtained by adding a suppression term to the objective weighted by the positive parameter μ:

$$\min_{x \in S} f(x) + \mu h' |x|_*. \tag{19}$$

Here h is a nonnegative vector in R^ℓ.

One approach for solving (19) utilizes the concave exponential approximation (11) of the step function. The resulting problem is the minimization of a concave function bounded below over a polyhedral region not having straight lines going to infinity in both directions. This concave minimization problem has a vertex solution [137, Corollaries 32.3.3 and 32.3.4]. The successive linearization approach computes a locally optimal solution to the concave problem by identifying vertices of the feasible region via a sequence of linear programs.

The successive linearization approach for obtaining minimum support solutions of (18) was introduced in [103] for a differentiable f utilizing the negative exponential

(11) to approximate the step function. The approach was generalized in [107] for a nondifferentiable f using its supergradient (2). Existence of a vertex solution for a finite smoothing parameter to the problem with differentiable f and negative exponential which also solves (19) was established in [25]. The result is generalized for nondifferentiable f in [106].

We restate [25, Theorem 2.1].

Theorem 2.1.2 Existence of Exact Vertex Solution for Finite Value of Smoothing Parameter *Let $f : R^\ell \to R$ be bounded below on the polyhedral set S that contains no straight lines going to infinity in both directions, let f be concave on R^ℓ, let μ be a fixed positive number and h be a nonnegative vector in R^ℓ. Then for a sufficiently large positive but finite value α_0 of α, the smooth problem*

$$\min_{(s,z)\in T} f(s) + \mu h'(e - \varepsilon^{-\alpha z}),\ (s,z) \in T := \{(s,z) \mid s \in S, -z \leq s \leq z\}, \tag{20}$$

has a vertex solution that also solves the original nonsmooth problem

$$\min_{s\in S} f(s) + \mu h'|s|_*. \tag{21}$$

Proof Since the objective function of (20) is concave on (s,z) in $R^{2\ell}$ and is bounded below on T, it follows by [137, Corollaries 32.3.3 and 32.3.4] that it has a vertex $(s(\alpha), z(\alpha))$ of T as a solution for each $\alpha > 0$. Since T has a finite number of vertices, one vertex, say $(\bar{s}, \bar{z})$, will repeatedly solve problem (20) for some sequence

$\{\alpha_0, \alpha_1, \ldots\} \uparrow \infty$. Hence for $\alpha_i \geq \alpha_0$,

$$\begin{aligned}
f(\bar{s}) + \mu h'(e - \varepsilon^{-\alpha_i \bar{z}}) &= f(s(\alpha_i)) + \mu h'(e - \varepsilon^{-\alpha_i z(\alpha_i)}) \\
&= \min_{(s,z)\in T} f(s) + \mu h'(e - \varepsilon^{-\alpha_i z}) \\
&= \min_{s\in S} f(s) + \mu h'(e - \varepsilon^{-\alpha_i |s|}) \\
&\leq \inf_{s\in S} f(s) + \mu h'|s|_*,
\end{aligned} \tag{22}$$

where the last inequality follows from

$$h'|s|_* \geq h'(e - \varepsilon^{-\alpha|s|}), \forall s \in R^\ell. \tag{23}$$

Letting $i \longrightarrow \infty$ it follows that

$$f(\bar{s}) + \mu h'|\bar{s}|_* = \lim_{i\longrightarrow\infty} f(\bar{s}) + \mu h'(e - \varepsilon^{-\alpha_i \bar{z}}) \leq \inf_{s\in S} f(s) + \mu h'|s|_*. \tag{24}$$

Since $\bar{s} \in S$, it follows that $\bar{s}$ solves (21). Since $(\bar{s}, \bar{z})$ is a vertex of T, it follows that $\bar{s}$ is a vertex of S. $\square$

By making the following identifications,

$$\ell = n + 1 + m + k,$$

$$s = [w' \;\; \gamma \;\; y' \;\; z']',$$

$$S = \left\{ (w, \gamma, y, z) \;\middle|\; \begin{array}{c} -Aw + e\gamma + e \leq y, \\ Bw - e\gamma + e \leq z, \\ y \geq 0, z \geq 0 \end{array} \right\}, \tag{25}$$

$$h = [e' \, 0 \, 0' \, 0']',$$

$$f(s) = \tfrac{e'y}{m} + \tfrac{e'z}{k},$$

$$\mu = \tfrac{\lambda}{1-\lambda},$$

the fundamental feature selection (FS) problem (9), or equivalently (8), becomes a special case of problem (21) which we solve by its smooth version (20). With the identifications (25) this smooth version is equivalent to FSV (13).

Hence, by solving the FSV problem for sufficiently large but finite α, it follows from Theorem 2.1.2, that we also have computed a solution to the original discontinuous FS problem (9). The method which we propose for solving the FSV problem is the successive linearization approximation (SLA) method of minimizing a concave function on a polyhedral set which is a finitely terminating stepless Frank-Wolfe algorithm [62]. In [103] finite termination of the SLA was established for a differentiable concave function, and in [107] for a nondifferentiable concave function using its supergradient

(2). We now state the algorithm.

Algorithm 2.1.3 Successive Linearization Algorithm (SLA) for FSV (13). *Choose* $\lambda \in [0,1)$. *Start with a random* (w^0, γ^0). *Set* $y^0 = (-Aw^0 + e\gamma^0 + e)_+$, $z^0 = (Bw^0 - e\gamma^0 + e)_+$, $v^0 = |w^0|$. *Having* $(w^i, \gamma^i, y^i, z^i, v^i)$ *determine* $(w^{i+1}, \gamma^{i+1}, y^{i+1}, z^{i+1}\ v^{i+1})$ *by solving the linear program:*

$$(w^{i+1}, \gamma^{i+1}, y^{i+1}, z^{i+1}, v^{i+1}) \in$$
$$\arg\, vertex \min_{w,\gamma,y,z,v} \left\{ \begin{array}{c|c} & -Aw + e\gamma + e \leq y, \\ (1-\lambda)(\frac{e'y}{m} + \frac{e'z}{k}) & Bw - e\gamma + e \leq z, \\ +\lambda\alpha(\varepsilon^{-\alpha v^i})'(v - v^i) & y \geq 0,\ z \geq 0, \\ & -v \leq w \leq v \end{array} \right\} \tag{26}$$

Stop if $(w^i, \gamma^i, y^i, z^i, v^i)$ *is feasible and*

$$(1-\lambda)(\frac{e'(y^{i+1} - y^i)}{m} + \frac{e'(z^{i+1} - z^i)}{k}) + \lambda\alpha(\varepsilon^{-\alpha v^i})'(v^{i+1} - v^i) = 0 \tag{27}$$

We restate [103, Theorem 4.2] which establishes finite termination for Algorithm 2.1.3 at a stationary point, which may be a global solution as well. We note that in [107] a minimum principle necessary optimality condition is defined for minimizing a nondifferentiable concave function over a polyhedral set and finite termination by a similar successive linearization algorithm to such a point is established.

Theorem 2.1.4 SLA Finite Termination for FSV (13). *The iterates determined by (26) generate a strictly decreasing sequence of objective function values for the FSV* *problem (13) and terminate at an iteration $\bar{i}$ with a stationary point (which may also be a global minimum solution) that satisfies the following minimum principle necessary optimality criterion [98].*

$$(1-\lambda)(\frac{e'}{m}(y-y^{\bar{i}})+\frac{e'}{k}(z-z^{\bar{i}}))+\lambda\alpha(\varepsilon^{-\alpha v^{\bar{i}}})'(v-v^{\bar{i}})\geq 0, \quad \forall \text{ feasible } (w,\gamma,y,z,v). \tag{28}$$

We solve the FSV problem (13) by Algorithm 2.1.3.

We now turn our attention to solving the FSB problem (16).

Bilinear Algorithm

Due to the bilinear nature of the objective function of the FSB problem (16), we consider the fast, simple bilinear algorithm of [9, Algorithm 2.1]. We apply the algorithm to solve the FSB problem (16) as follows.

Algorithm 2.1.5 Bilinear Algorithm for FSB (16) *Choose $\lambda \in [0,1)$, $\mu \in (0,1)$. Start with any feasible (w^0, γ^0, y^0, z^0, v^0, r^0, u^0) to the FSB problem. Having (w^i, γ^i, y^i, z^i, v^i, r^i, u^i) determine the next iterate by solving two linear programs (the first can be solved in closed form):*

$$r^{i+1}\in \arg\min_{r}\{(1-\mu)\lambda e'r-\mu(v^i)'r \mid 0\leq r\leq e\}\supset\{(-(1-\mu)\lambda e+\mu v^i)_*\}, \tag{29}$$

$$
(w^{i+1}, \gamma^{i+1}, y^{i+1}, z^{i+1}, v^{i+1}, u^{i+1}) \in
$$

$$
\arg\, vertex \min_{w,\gamma,y,z,v,u} \left\{ \begin{array}{c|c} & -Aw + e\gamma + e \leq y, \\ (1-\mu)(1-\lambda)(\frac{e'y}{m} + \frac{e'z}{k}) & Bw - e\gamma + e \leq z, \\ & y \geq 0, z \geq 0, \\ +\mu(-(r^{i+1})'v + e'u) & -v \leq w \leq v \\ & u \geq 0,\ u - v \geq 0 \end{array} \right\}. \quad (30)
$$

Stop when:

$$
(1-\mu)\left[(1-\lambda)\left(\frac{e'}{m}(y^{i+1} - y^i) + \frac{e'}{k}(z^{i+1} - z^i)\right) + \lambda e'(r^{i+1} - r^i)\right] +
$$

$$
\mu\left[-(r^{i+1})'v^{i+1} + (r^i)'v^i + e'(u^{i+1} - u^i)\right] = 0. \quad (31)
$$

Note: For a fixed $\lambda \in [0, 1)$, the parameter μ is chosen as the smallest in $(0, 1)$ so that the following complementarity condition holds at termination:

$$
-(r^{i+1})'v^{i+1} + e'u^{i+1} = (r^{i+1})'(u^{i+1} - v^{i+1}) + (u^{i+1})'(r^{i+1} + e) = 0. \quad (32)
$$

OBD Adaptation

We consider the Optimal Brain Damage method [92] for reducing neural network complexity and adapt it to problem (6). Because the objective function of (6) is piecewise-linear, the second derivatives on which the OBD method is based do not exist. However, at a solution point $(\bar{w}, \bar{\gamma})$ of (6), or equivalently (7), the directional derivatives of $f(w, \gamma)$ of (6) in the directions of the components $\pm w_i$ are generally nonzero. Hence the we propose examining these directional derivatives in the $2n$ directions $(\pm w_1, \ldots, \pm w_n)$ at the solution point $(\bar{w}, \bar{\gamma})$, and suppress those features x_i corresponding to w_i for which the directional derivatives in the directions $\pm w_i$ are less than some tolerance, then re-solve the linear program (7) with these w_i set to zero. We summarize our adaptation of the OBD algorithm for problem (6) as follows.

Algorithm 2.1.6 Optimal Brain Damage (OBD) Algorithm for (6). *Choose small positive parameters δ_1 and δ_2.*

(i) *Solve (6) for $(\bar{w}, \bar{\gamma})$ or equivalently the RLP (7) for $(\bar{w}, \bar{\gamma}, \bar{y}, \bar{z})$.*

(ii) *Set $w_i = 0$ for $i \in I$ and re-solve (6) or (7) where:*

$$I := \{i \Big| \mid f(\bar{w}_1, \ldots, \bar{w}_{i-1}, \bar{w}_i \pm \delta_1, \bar{w}_{i+1}, \ldots, \bar{w}_n, \bar{\gamma}) - f(\bar{w}, \bar{\gamma}) \mid \leq \delta_2,\ i = 1, \ldots, n\},$$

where $f(w, \gamma)$ is the objective function of (6).

In the implementation of this OBD adaptation, we introduce a parameter $\lambda \in [0, 1)$ which has approximately the same effect as λ appearing in FSS (12), FSV (13) and FSB (16). Let $(\bar{w}, \bar{\gamma}, \bar{y}, \bar{z})$ be the solution of the RLP (7) in Step (i), above, and let κ be the absolute value of the maximal directional derivative for fixed δ_1 at the point $(\bar{w}, \bar{\gamma})$. Then δ_2 in Step (ii) is chosen as $\lambda \cdot \kappa$. Thus λ has the effect of suppressing all components of w when $\lambda = 1$ and suppressing no components when $\lambda = 0$.

This OBD adaptation is considered a wrapper model [83].

2.1.4 Numerical Results

We evaluated the proposed algorithms for solving the FSS problem (12), the FSV problem (13), the FSB problem (16), and the OBD adaptation (Algorithm 2.1.6), in comparison with solutions produced by the RLP (7).

The two datasets used are publicly available from the UCI Machine Learning Repository [119]. We describe the datasets next.

Datasets

The WPBC dataset as it exists in the UCI ML Repository consists of 198 instances corresponding to malignant breast cancer patients. Of these 198, 151 have not had a cancer recurrence whereas 47 have. Each instance is represented by 35 features.

The first feature is an identification number. The second feature is a flag in indicating an outcome of recur or non-recur. The third feature is an integer. In cases in which cancer has recurred [outcome = recur], the third feature is the number of months from time of malignant cancer diagnosis to the time cancer has recurred (TTR = time to recur). In cases in which cancer has not recurred [outcome = non-recur], the third feature is disease-free time in months (DFS). Features 4-33 are the mean, standard error and "worst" values of ten real-valued features computed for each cell nucleus by the **Xcyt** [154] program at time of diagnosis. Feature 34 is the diameter of the excised tumor and feature 35 is the number of positive axillary lymph nodes observed at time of surgery.

Four of the 198 instances are missing the value for feature 35. These instances were discarded. We created the sets $\mathcal{A}$ and $\mathcal{B}$ to address the following question: "Can we recognize instances in which cancer recurred within 24 months from instances where cancer did not recur within 24 months?" Hence, the remaining 194 instances were processed in the following way. Instances having [outcome = recur] and [time to recur $\leq$ 24 months] were placed in set $\mathcal{A}$. Instances having either [outcome = recur] or [outcome = non-recur] and [TTR/DFS $>$ 24 months] were placed in set $\mathcal{B}$. Instances with [outcome = non-recur] and [DFS $\leq$ 24 months] were removed from the dataset (i.e. examples representing patients not followed for more than 24 months). This processing procedure results in 147 instances, 28 in set $\mathcal{A}$ and 119 in set $\mathcal{B}$. The information used to assign instances to one of the two sets (features 2 and 3) are removed. The resulting sets $\mathcal{A}$ and $\mathcal{B}$ are disjoint point sets in R^{32}.

The sets $\mathcal{A}$ and $\mathcal{B}$ were augmented with 6 random features sampled from a normal distribution with mean 86.2851 and standard deviation 33.1157. The value of 86.2851 is the grand mean over the 32 original WPBC features and 33.1157 is the average standard deviation over these features. Hence when referring to the WPBC dataset, we refer to the disjoint point sets $\mathcal{A}$ and $\mathcal{B}$ in R^{38}.

The Johns Hopkins University Ionosphere dataset consists of radar data collected by a system in Goose Bay, Labrador. The dataset is split into two sets. The first set represents "good" radar returns, those showing evidence of some type of structure. The second set consists of "bad" radar returns, those that do not show evidence of structure; the signals pass through the ionosphere.

There are 351 instances with 35 features per instance. Feature 1 takes continuous values in $[0, 1]$. Feature 2 is zero for all instances. Features 3 through 34 take continuous values on $[-1, 1]$. The set $\mathcal{A}$ consists of features 1 through 34 corresponding to "good" radar returns. The set $\mathcal{B}$ consists of the same features corresponding to "bad" radar returns. Notice that feature 2 is irrelevant.

Similarly, we augmented this dataset by adding 6 random features sampled from a uniform distribution on $[-1, 1]$. When referring to the Ionosphere dataset, we are referring to the disjoint point sets $\mathcal{A}$ and $\mathcal{B}$ in R^{40}.

Numerical Experiments

We investigated the generalization ability of a classification function estimated by one of the feature selection methods (FSS (12), FSV (13), FSB (16) or OBD (Algorithm 2.1.6)) in comparison with the generalization ability of a classification function estimated by the RLP (7).

We compare generalization ability by the re-sampled paired t test [53]. This statistical test, which is most popular in the machine learning literature, consists of conducting 30 trials. In each trial, the available data is randomly divided into a training set $(A_{\text{Train}}, B_{\text{Train}})$, consisting of $\frac{2}{3}$ of the available data, and a testing set $(A_{\text{Test}}, B_{\text{Test}})$. One of the feature selection methods is applied to the training subset to produce a classifier. The value of the parameter λ was chosen as the maximal element in a set Λ having best generalization ability, measured by 5-fold cross-validation [152] on the training set. Similarly, a classifier is constructed by solving the robust linear program on data $(A_{\text{Train}}, B_{\text{Train}})$. Both are tested on the testing set $(A_{\text{Test}}, B_{\text{Test}})$ and the proportion of test set examples misclassified is recorded for each classifier, for each trial. The averages of these proportions, over the 30 trials, are then compared. Results are summarized below in Tables 1 and 2.

For each of FSS (12), FSV (13), FSB (16) and OBD (Algorithm 2.1.6), the candidate set of λ-values was $\Lambda := \{0.15, 0.20, \ldots, 0.95\}$. When solving the FSB problem (16) by Algorithm 2.1.5 for a fixed value of λ, the parameter μ was chosen as the smallest in $\{0.05, 0.15, 0.25, \ldots, 0.95\}$ so that the complementarity condition (32) holds at termination. When solving the FSS problem (12) with the MINOS package [120], the maximum iteration limit was set to 10,000.

The statistical software package MINITAB [116] was used to calculate p-values.

Numerical Results

Table 1 summarizes test set results and Table 2 summarizes the number of features utilized on the WPBC dataset. Table 3 summarizes test set results and Table 4 summarizes the number of features utilized on the Ionosphere dataset. $AVE[err]$ (Tables 1 and 3) is the average proportion of test set points misclassified by the estimator constructed by the given algorithm over the 30 trials. The p-value indicates statistical significance in the average test error between the given feature selection approach and the RLP (7) [1]. $AVE[|w|_*]$ (Tables 2 and 4) is the average number of nonzero elements in the vector w computed by the given algorithm over the 30 trials.

Table 1: Average test set results on the WPBC dataset.

		$AVE[err]$	p-value (vs. RLP (7))
Sigmoid	FSS (12)	0.293 ± 0.062	0.025
Concave	FSV (13)	**0.274** ± 0.062	**0.003**
Bilinear	FSB (16)	0.362 ± 0.067	0.046
Brain	OBD (Alg. 2.1.6)	0.297 ± 0.068	0.085
Robust LP	RLP (7)	0.330 ± 0.067	

$AVE[err]$ = average test set error.
Bold indicates best.
Value following "±" is one standard deviation.

[1] Specifically, this is the p-value of a two-tailed paired t-test testing the null hypothesis that the difference in "Test" errors for the feature selection and RLP classifiers is zero [52].

Table 2: Average number of features used on the WPBC dataset.

		$AVE[\|w\|_*]$
Sigmoid	FSS (12)	6.53 ± 3.03
Concave	FSV (13)	3.77 ± 2.64
Bilinear	FSB (16)	12.13 ± 4.131
Brain	OBD (Alg. 2.1.6)	4.80 ± 5.13
Robust LP	RLP (7)	32.73 ± 4.03

$AVE[|w|_*]$ = average # of features used.
Value following "±" is one standard deviation.

WPBC Results

We immediately note that even though there are 38 original problem features in the WPBC dataset, the classifiers obtained by solving the RLP (7) over the 30 trials utilized on average only 32.73 of these features without a feature selection term (see Table 2).

Results in Table 1 indicate that the classifiers produced by solving the FSV problem (13) improve average test set error by 16.9% over classifiers computed by solving the RLP (7). The p-value of 0.003 indicates that the difference in test set error is significant. FSV reduced the average number of problem features utilized by 88.5% (see Table 2).

The classifiers produced by solving the FSS problem (12) improved generalization ability on the WPBC dataset by 11.1% (see Table 1). The small p-value indicates that this is significant. The FSS classifiers reduced the number of features by 80.0% over the classifiers constructed by the RLP (7) (see Table 2).

The classifiers constructed by the OBD Algorithm 2.1.6 reduced test set error by 10.1% and the p-value indicates that this difference is significant at any confidence level below 1-0.085 = 0.915 (see Table 1). Number of features were reduced by 85.3%

(Table 2).

When interpreting the results of the classifiers produced by solving the FSB problem (16), it should be noted that the complementarity condition (32) was satisfied in only 18 of the 30 trials. Recall that the complementarity condition must be satisfied to correctly model the step function. In the 12 trials in which complementarity was not achieved, the solutions were computed with $\mu = 0.95$ and the weighting defined by μ in the objective of the FSB problem (16) was *not* on the term forcing separation of the set $\mathcal{A}$ and $\mathcal{B}$, but on the complementarity penalty term, $-(r^{i+1})'v + e'u$. It is possible that poor classifiers were obtained in these 12 trials. Furthermore, in the 18 trials in which complementarity was achieved (i.e. the step function *was* modeled exactly), the average value of μ in computing the solution of the FSB problem (16) was 0.90. In these 18 trials, the objective of the FSB problem (16) was weighted to achieve complementarity and again, it is possible that poor classifiers were obtained. The poor performance of the classifiers obtained by solving the FSB problem (16) may be due to the difficulty in modeling the step function on the WPBC dataset.

Ionosphere Results

Table 3: Average test set results on the Ionosphere dataset.

		$AVE[err]$	p-value (vs. RLP (7))
Sigmoid	FSS (12)	0.165 ± 0.038	0.34
Concave	FSV (13)	**0.143** ± 0.029	**0.11**
Bilinear	FSB (16)	0.162 ± 0.053	0.60
Brain	OBD (Alg. 2.1.6)	0.178 ± 0.056	0.072
Robust LP	RLP (7)	0.157 ± 0.035	

$AVE[err]$ = average test set error.
Bold indicates best.
Value following "±" is one standard deviation.

Table 4: Average number of features used on the Ionosphere dataset.

		$AVE[\lvert w\rvert_*]$
Sigmoid	FSS (12)	3.80 ± 1.52
Concave	FSV (13)	4.23 ± 1.96
Bilinear	FSB (16)	23.40 ± 12.96
Brain	OBD (Alg. 2.1.6)	10.00 ± 9.82
Robust LP	RLP (7)	39.00 ± 0.00

$AVE[|w|_*]$ = average # of features used.
Value following "±" is one standard deviation.

Results in Table 3 indicate that classifiers constructed by solving the FSV problem (13) reduced average test set error by 8.9%. The p-value indicates that the difference in average test set error may be significant. FSV also reduced the feature space by 89.1% over the classifiers constructed by solving the RLP (7) (Table 4).

Given the small difference between the performance of the classifiers produced by solving the FSS problem (12) and classifiers produced by solving the RLP (7) coupled with the large p-value, the performance of these classifiers is not significantly different (Table 3). But notice that the FSS (12) classifiers reduced dimensionality by 90% (Table 4).

The average performance of classifiers produced by the solving the FSB problem (16) were slightly worse (a difference of 0.005) than the RLP (7), but the high p-value indicates that these results are not significantly different (Table 3). The average value of μ in the objective of the FSB problem (16) over the 30 trials was 0.1033 and the complementarity penalty term, $-(r^{i+1})'v + e'u$, was zero at the solution in all trials. Hence, the step function was exactly modeled for a rather small value of μ and comparable separating surfaces were computed while reducing dimensionality by 40%

(Table 4).

The classifiers produced by the OBD Algorithm 2.1.6 performed worse than those constructed by the robust linear program. The p-value indicates that this difference may be significant (Table 3). It seems as though the OBD Algorithm fails to find "useful" features with which to separate the Ionosphere dataset.

We note that the maximum iteration limit of 10,000 in MINOS [120] was reached in numerous trials while solving the FSS problem (12) on the Ionosphere dataset.

The results on the WPBC and Ionosphere datasets indicated that classifiers computed via solving the FSV problem (13) improve most upon the generalization ability of those computed via the RLP (7) by reducing the number of original problem features utilized. FSV solutions were easily computed by Algorithm 2.1.3 by solving a finite sequence of linear programs (typically 5-12). We claim that the FSV approach is a viable data mining tool addressing the combinatorial feature selection problem arising in the classification task.

Note that Algorithm 2.1.3 computes a solution which is locally optimal. We cannot guarantee a solution which is globally optimal.

We propose initializing Algorithm 2.1.3 by choosing an initial *random* separating plane. This strategy produced "useful" separating planes defining our classification functions. The issue of initializing Algorithm 2.1.3 is open.

We now focus on the support vector machine (SVM) approach to classification.

2.2 Support Vector Machines for Classification

The support vector machine approach to classification was motivated by developments in statistical learning theory [157, 158, 160, 156, 161]. The principal motivation is a probabilistic bound on the generalization error [160] of a classifier consisting of the average error of the classifier over a finite training set plus a term dependent upon the *VC-Dimension* [161, 33] of the particular set of classification functions considered.

We are interested in the set of classification functions defined by separating planes, $w'x = \gamma$ on R^n. For functions $\hat{g}(x) = (w'x - \gamma)_*$ which correctly classify the training data (training error = 0) an upper bound on the VC-Dimension is available in terms of the training data and the norm of the weight vector $w \in R^n$ [143, 161, Proposition 2.1.1]. Hence by minimizing the norm of w, one decreases the bound on the VC-Dimension in the hope of decreasing the bound on generalization error.

Minimizing the norm of w is also intuitively plausible in the following sense. In the case of linearly separable training data, we can always compute *bounding planes* for the sets $\mathcal{A}$ and $\mathcal{B}$. The plane $w'x = \gamma + 1$ is a *bounding plane for* $\mathcal{A}$ since it can be determined so that $\mathcal{A} \subset \{x \in R^n \mid w'x \geq \gamma + 1\}$. Similarly, the plane $w'x = \gamma - 1$ is a *bounding plane for* $\mathcal{B}$ since it can be computed so that $\mathcal{B} \subset \{x \in R^n \mid x'w \leq \gamma - 1\}$. The *margin*, defined as the distance between these two bounding planes, measured by some norm $\|\cdot\|$ on R^n is precisely $\frac{2}{\|w\|'}$, where $\|\cdot\|'$ is the dual norm of $\|\cdot\|$ [104, Theorem 2.2]. See Figure 4.

Hence minimizing $\|w\|'$ is equivalent to computing a separating plane with maximum margin $\frac{2}{\|w\|'}$. Maximizing the margin of separation between the sets $\mathcal{A}$ and $\mathcal{B}$ intuitively should improve the generalization ability of the estimated classification

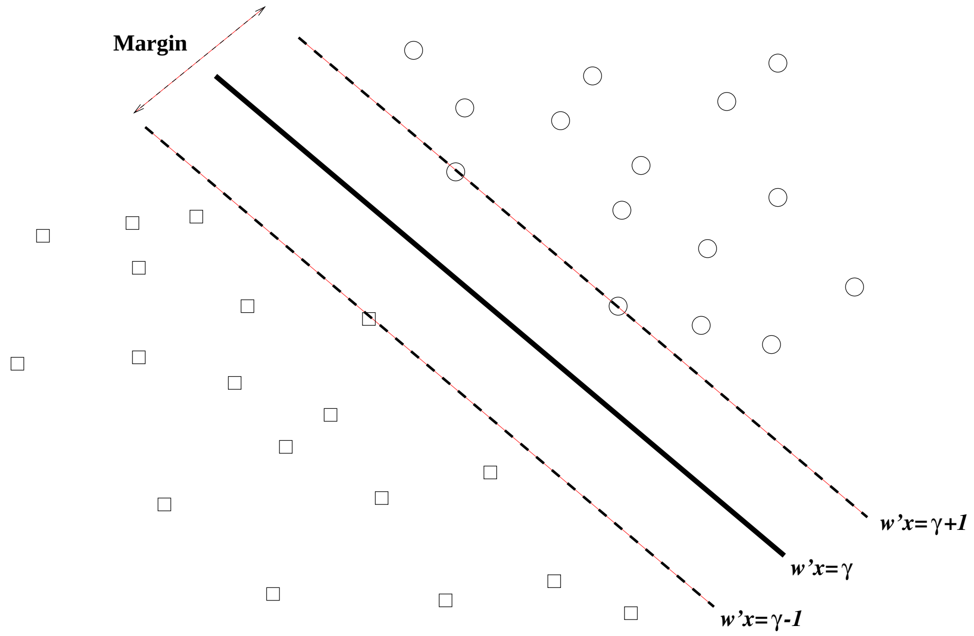


Figure 4: Elements of $\mathcal{A} = \bigcirc$. Elements of $\mathcal{B} = \square$.

function $\hat{g}(x) = (w'x - \gamma)_*$.

In the case where the training data are linearly inseparable (training error > 0), one attempts to minimize the separation error and the norm of w simultaneously.

Applications of support vector machine classifiers include isolated handwritten digit recognition [46, 144, 145, 34], object recognition [12], face detection [127] and text categorization [82]. The support vector machine has been extended beyond classification tasks to regression problems [149, 159, 54] and principle component analysis (PCA) [143, 146].

2.2.1 SVM Mathematical Programs

The support vector machine classifier is obtained by solving an optimization problem with an objective function which balances a term forcing separation between $\mathcal{A}$ and $\mathcal{B}$

and a term maximizing the margin of separation or equivalently, minimizing the norm of w.

This results in the following mathematical programming formulation of the SVM problem:

$$\min_{w,\gamma,y,z} \left\{ (1-\lambda)(e'y + e'z) + \frac{\lambda}{2}\|w\|' \;\middle|\; \begin{array}{c} -Aw + e\gamma + e \leq y, \\ Bw - e\gamma + e \leq z, \\ y \geq 0, z \geq 0, \end{array} \right\}, \lambda \in (0,1). \tag{33}$$

Points $A_i \in \mathcal{A}$ and $B_i \in \mathcal{B}$ corresponding to active constraints of (33) at optimality with positive dual variables constitute the *support vectors* of the problem. Support vectors from $\mathcal{A}$ are in the halfspace $\{x \in R^n \mid w'x \leq \gamma + 1\}$ (i.e. those points of $\mathcal{A}$ "on" or "below" the bounding plane $w'x = \gamma + 1$). Support vectors from $\mathcal{B}$ are in the halfspace $\{x \in R^n \mid w'x \geq \gamma - 1\}$ (i.e. those points of $\mathcal{B}$ "on" or "above" the plane $w'x = \gamma - 1$). These points are the *only* data points that are relevant in determining the optimal separating plane in the following sense: re-solving (33) over the support vectors *only* produces the same separating plane. The number of support vectors is usually small and is also proportional to a bound on the generalization error of the classifier [127]. See Figure 5.

We next formulate the support vector machine classification problem utilizing different norms to measure the margin or distance between the bounding planes $w'x = \gamma + 1$ and $w'x = \gamma - 1$. We refer to the various formulations as SVM-i, where the i-norm of w is minimized and hence the dual norm to the i-norm is used measure the margin between the bounding planes.

If the 1-norm is used to measure the distance between the planes, then the dual

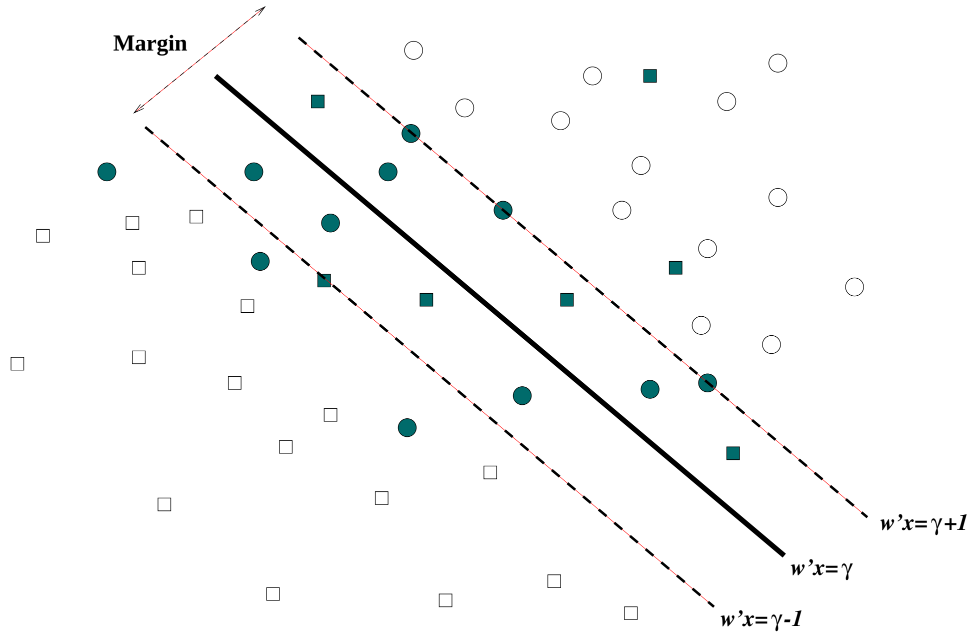


Figure 5: Elements of $\mathcal{A} = \bigcirc$. Elements of $\mathcal{B} = \square$. Shaded elements are the *support vectors.*

to this norm is the ∞-norm and accordingly $\|w\|' = \|w\|_\infty$ in (33) which leads to the following formulation:

$$\min_{w,\gamma,y,z} \left\{ (1-\lambda)(e'y + e'z) + \frac{\lambda}{2}\|w\|_\infty \;\middle|\; \begin{array}{c} -Aw + e\gamma + e \le y, \\ Bw - e\gamma + e \le z, \\ y \ge 0, z \ge 0. \end{array} \right\}, \lambda \in (0,1).$$

The above problem is equivalent to the following linear programming formulation:

$$\text{(SVM-}\infty\text{)}\ \min_{w,\gamma,y,z,\nu} \left\{ (1-\lambda)(e'y+e'z)+\frac{\lambda}{2}\nu \;\middle|\; \begin{array}{c} -Aw+e\gamma+e \le y, \\ Bw-e\gamma+e \le z, \\ y \ge 0, z \ge 0, \\ -e\nu \le w \le e\nu. \end{array} \right\},$$

$$\lambda \in (0,1). \quad (34)$$

Similarly if we use the ∞-norm to measure the distance between the planes, then the dual to this norm is the 1-norm and accordingly $\|w\|' = \|w\|_1$ in (33) which leads to the following formulation:

$$\min_{w,\gamma,y,z} \left\{ (1-\lambda)(e'y+e'z)+\frac{\lambda}{2}\|w\|_1 \;\middle|\; \begin{array}{c} -Aw+e\gamma+e \le y, \\ Bw-e\gamma+e \le z, \\ y \ge 0, z \ge 0. \end{array} \right\},\ \lambda \in (0,1).$$

The above problem is equivalent to the following linear programming formulation:

$$\text{(SVM-1)}\ \min_{w,\gamma,y,z,s} \left\{ (1-\lambda)(e'y+e'z)+\frac{\lambda}{2}e's \;\middle|\; \begin{array}{c} -Aw+e\gamma+e \le y, \\ Bw-e\gamma+e \le z, \\ y \ge 0, z \ge 0, \\ -s \le w \le s. \end{array} \right\},$$

$$\lambda \in (0,1). \quad (35)$$

We note that an optimization formulation was proposed and implemented in [97] for computing a separating plane by forcing the bounding planes to be as far apart as possible.

Usually the support vector machine problem is formulated using the 2-norm in the objective [161, 6]. Since the 2-norm is dual to itself, it follows that the margin is also measured in the 2-norm when this formulation is used. In this case $\|w\|' = \|w\|_2$, and one usually appends the term $\frac{\lambda}{2}\|w\|_2^2$ to the objective of (33) resulting in the following quadratic program:

$$\text{(SVM-2)} \min_{w,\gamma,y,z} \left\{ (1-\lambda)(e'y+e'z)+\frac{\lambda}{2}w'w \;\middle|\; \begin{array}{c} -Aw+e\gamma+e\le y, \\ Bw-e\gamma+e\le z, \\ y\ge 0, z\ge 0. \end{array} \right\}, \qquad \lambda\in(0,1). \quad (36)$$

We note the following regarding the use of $\|w\|_2^2$ instead of $\|w\|_2$ in (36) which renders the problem much easier to solve. It turns out that the formulations are equivalent for values of $\lambda\in(0,\bar\lambda]$ for some $\bar\lambda\in(0,1)$, as we show now.

Let $\bar X$ denote the solution set of the following linear program:

$$\bar X := \arg\min_{w,\gamma,y,z} \left\{ e'y+e'z \;\middle|\; \begin{array}{c} -Aw+e\gamma+e\le y, \\ Bw-e\gamma+e\le z, \\ y\ge 0, z\ge 0. \end{array} \right\}.$$

Then, for some $\bar\lambda\in(0,1)$, it follows from [108] that a solution of (36) is a solution to the following problem:

$$\min_{w,\gamma,y,z\in\bar X} \frac{1}{2}\|w\|_2^2,$$

as well as of

$$\min_{w,\gamma,y,z\in\bar{X}} \frac{1}{2}\|w\|_2.$$

Nonlinear separating surfaces, which are linear in their parameters, can also easily be handled by the formulations (7), (12), (13), (16), (34) and (35) [96]. If the data are mapped nonlinearly via $\Phi : R^n \to R^\ell$, a nonlinear separating surface in R^n is easily computed as a linear separator in R^ℓ. In practice, one usually solves (36) by way of its dual [98]. In this formulation, the data enter only as inner products which are computed in the *transformed* space via a kernel function $K(x, y) = \Phi(x) \cdot \Phi(y)$ [33, 161, 164].

We note that separation errors in (34) - (36) are weighted equally conforming to the SVM formulations in [33, 161]. In contrast, the formulations (7), (12), (13), (16) measure *average* separation error. Minimizing average separation error in (7) ensures that the solution $w = 0$ occurs iff $\frac{e'A}{m} = \frac{e'B}{k}$, in which case it is not unique [8, Theorem 2.5].

2.2.2 Numerical Comparison with FSV and RLP

Numerical tests were conducted to investigate feature suppression and generalization ability of the support vector machine approaches (34) – (36), the concave minimization FSV problem (13) and the robust linear program (7) [22].

The six publicly available datasets used in this comparison are discussed next.

Datasets

We used two variants of the Wisconsin Prognostic Breast Cancer Database (see Section 2.1.4). The first set was created where the elements of the set $\mathcal{A}$ were 30 nuclear features plus diameter of excised tumor and number of positive lymph nodes of instances corresponding to patients in which cancer had recurred in less than 24 months (28 points). The set $\mathcal{B}$ consisted of the same features for patients in which cancer had not recurred in less than 24 months (119 points). Data corresponding to patients that have not been followed for 24 months was discarded.

The second variant of the data set consisted of the same 32 features, but but splits the data into $\mathcal{A}$ and $\mathcal{B}$ differently. Elements of $\mathcal{A}$ corresponds to patients with a cancer recurrence in less than 60 months (41 points) and $\mathcal{B}$ corresponds to patients which cancer had not recurred in less than 60 months (69 points). Data representing patients not followed for 60 months was removed.

The Johns Hopkins University Ionosphere data set was also used (see Section 2.1.4). The set $\mathcal{A}$ consists of 225 elements in R^{34}. The set $\mathcal{B}$ consists of 126 elements in R^{34}.

The Cleveland Heart Disease data set is available from [119] and the version used here was obtained by removing the six examples with missing feature values from the file processesed.cleveland.data. The resulting 297 examples were then divided into the sets $\mathcal{A}$ and $\mathcal{B}$ using values of the 14th feature. This feature is an integer between 0 and 4. A value of zero indicates no heart disease present and nonzero values indicate a presence of heart disease. Set $\mathcal{A}$ consists of features 1 - 13 for instances with corresponding feature 14 values of 0 or 1. Set $\mathcal{A}$ has 214 elements. The set $\mathcal{B}$ consists of features 1 - 13 for instances with corresponding feature 14 values of 2, 3 or 4. Set $\mathcal{B}$ has 83

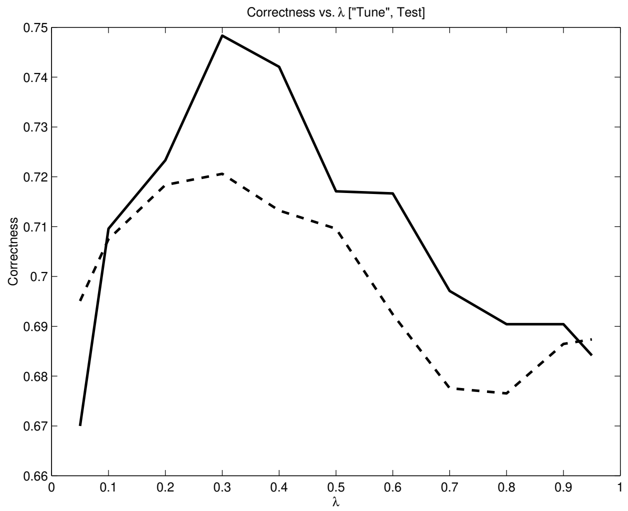


Figure 6: Tuning and testing sets correctnesses versus λ. **Dashed** = "tuning" correctness, **Solid** = test correctness. Classifier computed via SVM-1 (35).

elements.

The Pima Indians Diabetes data set consists of 768 instances with 8 features plus a class label [119]. The 500 instances with class label "0" were place in $\mathcal{A}$, the 268 instances with class label "1" were placed in $\mathcal{B}$.

The BUPA Liver Disorders data set consists of 345 instances with 6 features plus a selector field used to split the data into 2 sets (see documentation [119]). Set $\mathcal{A}$ consists of 145 instances, the set $\mathcal{B}$ consists of 200 instances.

Numerical Experiments

Our goal was to evaluate the generalization ability of the classifiers obtained by solving: the concave minimization problem FSV (13), SVM 1-norm problem (35), the SVM ∞-norm problem (34), the SVM 2-norm problem (36) and the robust linear program (RLP) (7). We estimate the generalization ability of a classifier via 10-fold cross-validation [152].

We note that the objective function parameter λ, which can induce sparsity, must be chosen carefully to maximize the generalization ability of the resulting classifier. We employ the following "tuning set" procedure for choosing λ at each fold of 10-fold cross-validation: For each λ in a candidate set Λ, we perform the following:

1. Set aside 10% of the training data as a "tuning" set.
2. Obtain a classifier for the given value of λ.
3. Determine correctness on the "tuning" set.
4. Repeat steps 1 – 3 ten times, each time setting aside a different 10% portion of the training data.

The value of λ is fixed as that with maximum average tuning correctness (ties are broken by choosing the smallest λ-value). This is the value used for the given fold of 10-fold cross-validation. The set Λ is a set of candidate values and for these experiments was set at: $\Lambda = \{0.05, 0.10, 0.20, \ldots, 0.90, 0.95\}$. The curves in Figure 6 indicate that the value of λ that maximizes the "tuning" score (dashed curve in Figure 6) is a good estimate of the value of λ that maximizes the test set correctness (solid curve).

Numerical Results

Table 5 summarizes the average number of original problem features selected by the classifiers trained by each of the methods.

Table 6 summarizes the results of the 10-fold cross-validation experiments on the 6 real-world data sets. All "Train" and "Test" numbers presented are average error over 10-folds. The p-value is an indicator of significance difference in "Test" error between the classifiers obtained by solving FSV (13) and the classifiers obtained by solving the SVM 1-norm problem (35) [2].

Table 5: Average number of features selected over ten-fold cross-validation.

Dataset	FSV (13)	SVM-1 (35)	SVM-∞ (34)	SVM-2 (36)	RLP (7)
WPBC-24	**3.9** ± 2.0	5.4 ± 1.5	32 ± 0	32 ± 0	32 ± 0
WPBC-60	**2.6** ± 2.4	4.3 ± 2.5	32 ± 0	32 ± 0	32 ± 0
Ionosphere	**10.4** ± 1.8	11.1 ± 3.6	34 ± 0	33 ± 0	33 ± 0
Cleveland	**6.4** ± 2.2	9.3 ± 1.8	13 ± 0	13 ± 0	13 ± 0
Pima	**5.3** ± 3.2	6.0 ± 1.5	8 ± 0	8 ± 0	8 ± 0
BUPA	**4.5** ± 1.3	5.8 ± 0.4	6 ± 0	6 ± 0	6 ± 0

Bold indicates fewest number of features utilized.
Values following "±" are 1 standard deviation.

Table 6: FSV, SVM-i, RLP: Ten-fold cross-validation test results (errors).

Data Set	FSV (13)	SVM-1(35)	p-Value	SVM-∞ (34)	SVM-2 (36)	RLP (7)
WPBC-24	0.336 ± 0.111	0.289 ± 0.117	0.125	0.290 ± 0.102	0.245 ± 0.100	0.339 ± 0.111
WPBC-60	0.330 ± 0.130	0.338 ± 0.109	0.641	0.336 ± 0.120	0.338 ± 0.083	0.365 ± 0.099
Ionosphere	0.159 ± 0.060	0.139 ± 0.063	0.125	0.159 ± 0.035	0.140 ± 0.048	0.140 ± 0.049
Cleveland	0.191 ± 0.096	0.155 ± 0.057	0.182	0.175 ± 0.070	0.165 ± 0.083	0.161 ± 0.067
Pima	0.254 ± 0.072	0.255 ± 0.065	0.889	0.250 ± 0.058	0.234 ± 0.048	0.262 ± 0.066
BUPA	0.348 ± 0.082	0.360 ± 0.086	0.170	0.354 ± 0.070	0.336 ± 0.075	0.357 ± 0.069

Values following "±" are 1 standard deviation on average test set error.

The FSV (13) and the SVM 1-norm (35) problems were the only approaches exhibiting feature selection (Table 5). On the six data sets tested, the FSV classifiers

[2] Specifically, this is the p-value of a two-tailed paired t-test testing the null hypothesis that the difference in "Test" errors for the FSV and SVM 1-norm classifiers is zero [52].

had a smaller test set error than the SVM-1 classifiers on three of the datasets and vice-versa on the remaining three (Table 6). The minimum p-value is 0.1246 indicating that classifiers obtained by the FSV (13) and the SVM 1-norm (35) approaches have similar generalization properties. The p-values for all pairwise comparisons between the various classification methods were computed for each of the six datasets. Two comparisons reject the hypothesis that average testing set errors are equal at the 95% confidence level. Both occur on the WPBC 24 month dataset. The first is between the SVM-2 (36) classifiers and the FSV (13) classifiers with p-value = 0.0316. The second is between the SVM-2 classifiers and the RLP (7) classifiers with p-value = 0.0071.

Note that applying the paired t-test to 10-fold cross validation results may indicate a difference in the average test set correctness when one is not present [53]. Thus the results of these experiments may be more similar than indicated by the p-values.

We note that the classifiers obtained by solving the SVM ∞-norm (34) suppressed none of the original problem features for all but the largest values of λ (near 1.0), which in general is of little use because it is often accompanied by poor separation. Similar behavior was observed when solving the SVM 2-norm (36) problem. Note that the ∞-norm is sensitive to outliers, as is the 2-norm squared.

The classifiers obtained by solving the FSV problem (13) selected fewer problem features than the any of the SVM formulations (34), (35), (36) and the RLP (7) (in which no feature suppression is present in the problem formulation). The FSV classifiers reduced the number of features used over SVM 1-norm by as much as 39.5% (WPBC 60 month), while maintaining comparable generalization performance. The FSV classifiers reduced the number of features utilized by as much as 91% over the SVM-∞, SVM-2 and RLP classifiers.

On the WPBC 24 month dataset, both the FSV classifiers (13) and the SVM 1-norm classifiers (35) most often selected a nuclear area feature and number of lymph nodes removed from the patient. These features are deemed relevant to the prognosis problem.

All linear programs formulations were solved using the CPLEX package [48] called from within MATLAB [110]. The quadratic programming problem (36) was solved using the MINOS optimization package [120] called from the GAMS modeling environment [31].

Table 7 summarizes average solve times for the Ionosphere dataset, the largest in this numerical comparison.

Table 7: Average running times: Ionosphere data set.

Method	Time (Seconds)
FSV (13)	30.94
SVM-1 (35)	3.09
SVM-∞ (34)	1.42
SVM-2 (36)	19.36
RLP (7)	1.28

Recall the FSV problem (13) is solved via the Successive Linearization Algorithm 2.1.3 and typically terminated after 5 - 12 linear programs. The SVM-1 (35), SVM-∞ (34) and the RLP (7) classifiers are obtained by simply solving a single linear program. The SVM-2 (36) classifiers require the solution of a quadratic program.

The numerical comparisons between the FSV feature selection approach, the support vector machine approaches and the robust linear program RLP indicate that classifiers computed by FSV have comparable generalization ability (as indicated by the p-values) yet utilize the fewest number of problem features. These results again indicate the utility of the FSV approach to the important data mining problem of classification.

Next, we consider an approach for solving general linear programs with a massive number of constraints [24]. This approach effectively scales the linear-programming-based classification formulations and makes them useful in large-scale KDD applications (see Section 1.3.1).

2.3 Massive Datasets

The approach discussed is a method for solving linear programs with a massive number of constraints. The basis of the approach lies in solving a succession of sufficiently small linear programs, resulting in a monotonic, finite algorithm converging to the solution of the full, massive linear program.

2.3.1 Linear Program Chunking (LPC) Algorithm

We consider a general linear program

$$\min_{x} \{c'x \,|Hx \geq b\}, \tag{37}$$

where $c \in R^n$, $H \in R^{m\times n}$ and $b \in R^m$. We state now our chunking algorithm [24, Algorithm 3.1] and establish its finite termination [24, Theorem 3.2] for the linear program (37), where m may be orders of magnitude larger than n. In its dual form our algorithm can be interpreted as a block-column generation method related to column generation methods of Gilmore-Gomory [68], Dantzig-Wolfe [49], [44, pp 198-200,428-429] and others [122, pp 243-248].

Algorithm 2.3.1 LPC: Linear Programming Chunking Algorithm for (37) *Let $[H \quad b]$ be partitioned into ℓ blocks, possibly of different sizes, as follows:*

$$\begin{bmatrix} H & b \end{bmatrix} = \begin{bmatrix} H^1 & b^1 \\ \vdots & \vdots \\ H^\ell & b^\ell \end{bmatrix}.$$

Assume that (37) and all subproblems (38) below, have vertex solutions.

At iteration $j = 1, \ldots$ compute x^j by solving the following linear program:

$$x^j \in arg\ vertex\ min \left\{ c'x \;\middle|\; \begin{array}{lcl} H^{(j\ mod\,\ell)}\, x & \geq & b^{(j\ mod\,\ell)} \\ \bar{H}^{(j\ mod\,\ell)-1}\, x & \geq & \bar{b}^{(j\ mod\,\ell)-1} \end{array} \right\}, \tag{38}$$

where $[\bar{H}^0 \quad \bar{b}^0]$ is empty and $[\bar{H}^j \quad \bar{b}^j]$ is the set of active constraints (that is all inequalities of (38) satisfied as equalities by x^j) with positive optimal Lagrange multipliers at iteration j.

Stop when $c'x^j = c'x^{j+\nu}$ for some input integer ν. Typically $\nu = 4$.

Theorem 2.3.2 Finite Termination of LPC Algorithm *The sequence $\{x^j\}$ generated by the LPC Algorithm 2.3.1 has the following properties:*

(i) The sequence $\{c'x^j\}$ of objective function values is nondecreasing and is bounded above by the global minimum of $\min_x \{c'x \,|Hx \geq b\}$.

(ii) The sequence of objective function values $\{c'x^j\}$ becomes constant, that is: $c'x^{j+1} = c'x^j$ for all $j \geq \bar{j}$ for some $\bar{j} \geq 1$.

(iii) For $j \geq \bar{j}$, active constraints of (38) at x^j with positive multipliers remain active for iteration $j+1$.

(iv) For all $j \geq \tilde{j}$ for some $\tilde{j} \geq \bar{j}$, x^j is a solution of the linear program (37) provided all active constraints at x^j have positive multipliers for $j \geq \bar{j}$.

This significant result is the basis of a fundamental computational approach for handling linear programs with massive constraints for which the subproblems (38) of the LPC Algorithm 2.3.1 have vertex solutions. To establish its validity we first prove a lemma [24, Lemma 3.3].

Lemma 2.3.3 *If $\bar{x}$ solves the linear program (37) and $(\bar{x}, \bar{u}) \in R^{n+m}$ is a Karush-Kuhn-Tucker (KKT) [98] point (i.e. a primal-dual optimal pair) such that $\bar{u}_I > 0$ where $I \subset \{1, \ldots, m\}$ and $\bar{u}_J = 0$, $J \subset \{1, \ldots, m\}$, $I \cup J = \{1, \ldots, m\}$, then*

$$\bar{x} \in arg\min\{c'x \,|H_I x \geq b_I\} \tag{39}$$

where H_I consists of rows H_i, $i \in I$ of H, and b_I consists of elements b_i, $i \in I$.

Proof The KKT conditions [98] for (37) satisfied by $(\bar{x}, \bar{u})$ are:

$$c = H'\bar{u},\ \bar{u} \geq 0,\ \bar{u}'(H\bar{x} - b) = 0,\ H\bar{x} - b \geq 0,$$

which under the condition $\bar{u}_I > 0$ imply that

$$H_I\bar{x} = b_I,\ \bar{u}_J = 0,\ H_J\bar{x} \geq b_J.$$

We claim now that $\bar{x}$ is also a solution of (39), because $(\bar{x}, \bar{u}_I)$ satisfy the KKT sufficient optimality conditions for (39):

$$c = {H_I}'\bar{u}_I,\ \bar{u}_I > 0,\ H_I\bar{x} = b_I.\square$$

Proof of Theorem 2.3.2

(i) By Lemma 2.3.3, $c'x^j$ is a lower bound for $c'x^{j+1}$. Hence the sequence $\{c'x^j\}$ is nondecreasing. Since the the constraints of (38) form a subset of of the constraints of (37), it follows that $c'x^j \leq \min_x \{c'x \,|Hx \geq b\}$.

(ii) Since there are a finite number of (feasible and infeasible) vertices to the original linear program (37) as well as of the subproblems (38), it follows that from a certain $\bar{j}$ onward, a finite subset of these vertices will repeat infinitely often. Since a repeated vertex gives the same value for $c'x$, it follows, by the nondecreasing property of $\{c'x^j\}$ just established, that all vertices between repeated vertices also have the same objective value $c'x$ and hence: $c'x^j = c'x^{j+1} \leq \min_x \{c'x \,|Hx \geq b\}, \ \forall j \geq \bar{j}$.

(iii) Let $\bar{j}$ be as defined in the theorem and let the index $t \in \{1, \ldots, m\}$ be that of some active constraint at iteration $\bar{j}$ with positive multiplier, that is: $H_t x^{\bar{j}} = b_t, \ u_t^{\bar{j}} > 0$, that has become inactive at the next step, that is: $H_t x^{\bar{j}+1} > b_t$. We then obtain the following contradiction by part (ii) above and the KKT saddlepoint condition:

$$0 = c'x^{\bar{j}+1} - c'x^{\bar{j}} \geq u^{\bar{j}\prime}(\bar{H}^{\bar{j}}x^{\bar{j}+1} - \bar{b}^{\bar{j}}) \geq u_t^{\bar{j}}(H_t x^{\bar{j}+1} - b_t) > 0.$$

(iv) By (ii) a finite number of vertices repeat infinitely for $j \geq \bar{j}$ all with constant $c'x^j$. Since active constraints with positive multipliers at iteration j remain active at iteration $j+1$ by (iii) and hence have a positive multiplier by assumption of part (iv), the set of active constraints with positive multipliers will remain constant for $j \geq \tilde{j}$ for some $\tilde{j} \geq \bar{j}$ (because there are a finite number of constraints) and hence x^j will remain a fixed vertex $\bar{x}$ for $j \geq \tilde{j}$. The point $\bar{x}$ will satisfy all the constraints of problem (37) because all constraints are eventually imposed on the infinitely repeated vertex $\bar{x}$. Hence $c'\bar{x}$ which lower-bounds the minimum of (37) is also a minimum of (37) because $\bar{x}$ is feasible. Hence the algorithm can be terminated at $j = \tilde{j}$. $\square$

Remark 2.3.4 *The possibility is not excluded (never observed computationally) that the objective function remains constant over a finite number of iterations then increases. Theorem 2.3.2 asserts that eventually the objective function will be constant and equal to the global minimum for all iterates* $j \geq \bar{j}$.

Remark 2.3.5 *In order to handle degenerate linear programs, i.e. those with active constraints with zero multipliers at a solution, we have modified the computational implementation of the LPC Algorithm 2.3.1 slightly by admitting into* $[\bar{H}^j \quad \bar{b}^j]$ *all active constraints at iteration* j *even if they have zero multipliers.*

Remark 2.3.6 *A practical implementation of the LPC algorithm was somewhat different from that given in Algorithm 2.3.1. The set of constraints forming any given subproblem consisted of the set of constraints with positive multipliers at the previous iteration plus* <u>*violated*</u> *constraints taken among those not considered in the previous iteration up to a prescribed maximum permissible number of constraints for each iteration. If the number of constraints with positive multipliers becomes larger than the maximum permissible number of constraints, then this maximum is made larger.*

Remark 2.3.7 *[An anti-cycling procedure, R. R. Meyer, [114]] A method for avoiding cycling without assuming nondegeneracy, suggested by R. R. Meyer [114], is the following. If the objective function does not increase at a given iteration of the LPC*

Algorithm 2.3.1, bring in at least one violated constraint, which is termed "permanent". A "permanent" constraint remains in all subsequent LPC subproblems (38) until the objective function strictly increases in which case all "permanent" constraints are removed. Specifically, if $c'x^{j-1} = c'x^j$, *then the set of constraints forming subproblem* $j+1$ *consist of* $[\bar{H}^j \quad \bar{b}^j]$ *(set of constraints at iteration* j *with positive multipliers) and the "permanent" constraints. If* $c'x^{i-1} < c'x^i$, $i > j$, *then the set of constraints forming subproblem* $i+1$ *consist of* $[\bar{H}^i \quad \bar{b}^i]$ *and the next corresponding "chunk" of constraints.*

Remark 2.3.8 *[A cycling example, R. R. Meyer, [114]]. The following example in which the inclusion of all active constraints, including those with zero multipliers, results in cycling of the LPC algorithm was suggested by R. R. Meyer [114]. This example is excluded by Assumption (iv) of Theorem 2.3.2 which requires all active constraints to be nondegenerate once the objective function becomes permanently a constant. Consider the following full linear program:*

$$\min_{x \in X} 0 \cdot x. \tag{40}$$

Here the full constraint set X *defines the feasible region depicted in Figure 7.*

Suppose the first LPC Algorithm 2.3.1 subproblem has feasible region depicted in Figure 8(a). Further assume that the solution to this subproblem occurs at the marked vertex. This subproblem solution has objective value equal to zero.

Suppose the second LPC Algorithm 2.3.1 subproblem has the feasible region depicted in Figure 8(b). We note that those constraints defining the "diamond" portion of the feasible region X *appear in both subproblems 1 and 2. Suppose the solution of this subproblem occurs at the marked vertex with objective value equal to zero.*

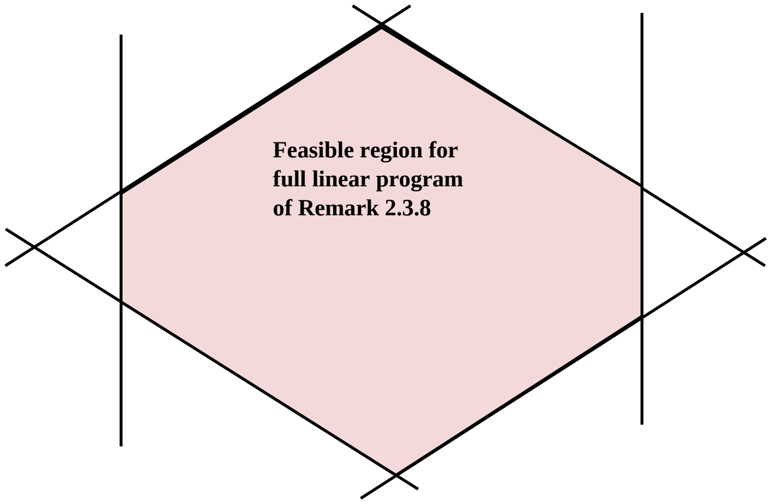


Figure 7: Feasible region X for the linear program (40).

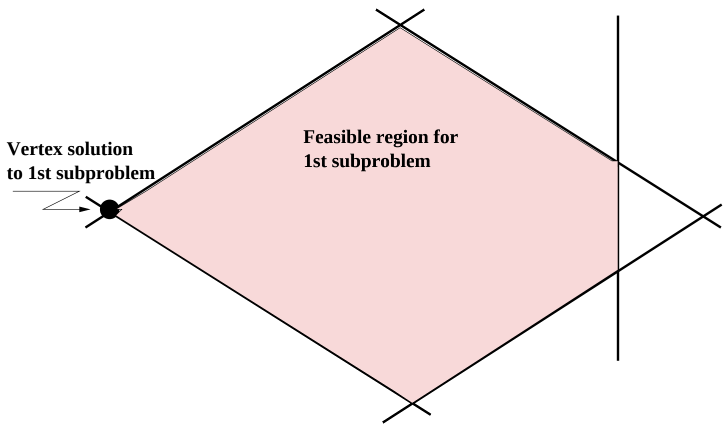
The LPC Algorithm 2.3.1 could cycle between these two subproblem solutions, neither of which are feasible for the full linear program (40) (see Figure 7).

The slight alteration of the LPC Algorithm 2.3.1 indicated in Remark 2.3.5 was motivated by the following counterexample [3]. Consider application of the RLP (7) to separate the following two sets $\mathcal{A}, \mathcal{B} \in R^2$, where:

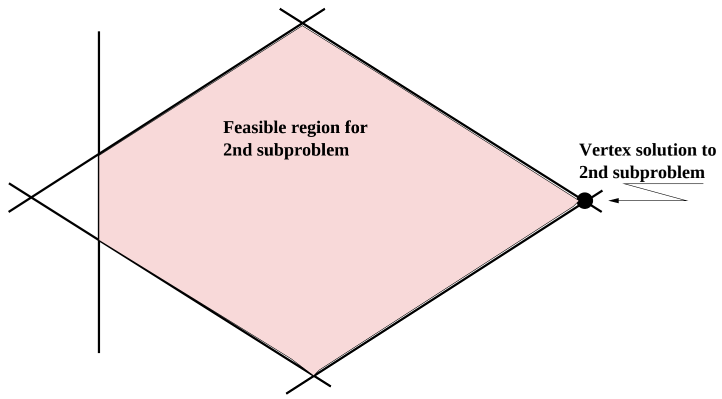
$$\begin{aligned} \mathcal{A} &:= \{(1,0),(0,1)\}, \\ \mathcal{B} &:= \{(0,0),(1,1)\}. \end{aligned} \tag{41}$$

The RLP (7) then is:

[3] Thanks to Dave Musicant.



(a) Feasible region for the first subproblem of the LPC Algorithm 2.3.1 applied to linear program (40).



(b) Feasible region for the second subproblem of the LPC Algorithm 2.3.1 applied to linear program (40).

Figure 8: Feasible regions for the LPC Algorithm 2.3.1 subproblems applied to linear program (40).

$$\min_{w_1,w_2,\gamma,y_1,y_2,z_1,z_2} \left\{ \frac{1}{2}\sum_{i=1}^{2} y_i + \frac{1}{2}\sum_{i=1}^{2} z_i \;\middle|\; \begin{array}{c} -1\cdot w_1 - 0\cdot w_2 + \gamma + 1 \le y_1, \\ 0\cdot w_1 - 1\cdot w_2 + \gamma + 1 \le y_2, \\ 0\cdot w_1 + 0\cdot w_2 - \gamma + 1 \le z_1, \\ 1\cdot w_1 + 1\cdot w_2 - \gamma + 1 \le z_2, \\ y_1, y_2, z_1, z_2 \ge 0. \end{array} \right\}. \tag{42}$$

Consider the 1st set of constraints:

$$\left\{ (w_1, w_2, \gamma, y_1, z_1) \;\middle|\; \begin{array}{c} -1\cdot w_1 - 0\cdot w_2 + \gamma + 1 \le y_1, \\ 0\cdot w_1 + 0\cdot w_2 - \gamma + 1 \le z_1, \\ y_1, z_1 \ge 0. \end{array} \right\}, \tag{43}$$

and the second set of constraints:

$$\left\{ (w_1, w_2, \gamma, y_2, z_2) \;\middle|\; \begin{array}{c} 0\cdot w_1 - 1\cdot w_2 + \gamma + 1 \le y_2, \\ 1\cdot w_1 + 1\cdot w_2 - \gamma + 1 \le z_2, \\ y_2, z_2 \ge 0. \end{array} \right\}. \tag{44}$$

The primal solution to the first subproblem with constraints (43) is $w_1 = 2$, $w_2 = 0$, $\gamma = 1$, $y_1 = 0$, $z_1 = 0$, with objective value is 0. The dual solution to the first subproblem is $u_1 = u_2 = u_3 = u_4 = 0$, even though the constraints $-1\cdot w_1 - 0\cdot w_2 + \gamma + 1 \le y_1$ and $0\cdot w_1 + 0\cdot w_2 - \gamma + 1 \le z_1$ are active. Since these constraints are degenerate, they do not get carried over into the second subproblem as originally prescribed by the LPC Algorithm 2.3.1. Hence the second subproblem is solved only over the constraints (44) yielding a primal solution: $w_1 = -2$, $w_2 = 1$, $\gamma = 0$, $y_2 = 0$, $z_2 = 0$. The primal objective of the second subproblem is 0 and the corresponding dual solution is $u_1 = u_2 = u_3 = u_4 = 0$. Hence the LPC Algorithm 2.3.1 will iterate between subproblems with constraints (43) and those with constraints (44). Subproblem objectives will always be zero hence the LPC Algorithm will terminate at a solution to one of these subproblems.

An optimal solution to (42) is $w_1 = -2$, $w_2 = 2$, $\gamma = 1$, $y_1 = 4$, $y_2 = z_1 = z_2 = 0$. The optimal objective to this full problem is 2. Note that if the set of constraints carried over from the first subproblem to the second included those that are active but degenerate, the optimal solution would have been easily computed via the LPC Algorithm.

We first discuss the application of the LPC Algorithm 2.3.1 to a general linear programming problem and then to massive discrimination problems.

2.3.2 Solving a General Linear Program: The WOODW Problem

The purpose of applying the LPC Algorithm 2.3.1 to a general linear program is principally to show that it can indeed handle such problems and that its usefulness is not merely confined to discrimination problems.

Upon the suggestion of Philip E. Gill [66], the LPC Algorithm 2.3.1 was applied to the WOODW linear programming problem. The WOODW problem data is publically available from the Netlib repository [1]. The primal linear program consists of 1085 equality constraints and 13 inequality constraints. There are 8405 variables. The optimal objective value is 1.3044763331. The constraints are 0.04% dense with 37474

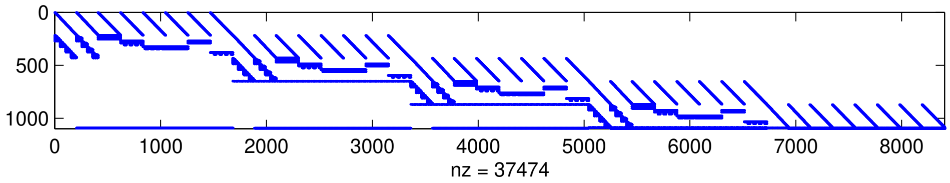


Figure 9: Nonzero elements of the WOODW primal constraint matrix.

nonzeros and the constraint matrix is structured (see Figure 9). The primal problem has the following form:

$$\min_{x} \{c'x \mid Ax = b, Bx \leq d, x \geq 0\}. \tag{45}$$

Since the LPC Algorithm 2.3.1 deals with linear programs with massive constraints, we applied it to the dual formulation [98] of (45) having 8405 constraints and 1098 variables:

$$\min_{u,v} \{-b'u + d'v \mid A'u - B'v \leq c, v \geq 0\}. \tag{46}$$

LPC Numerical Results for Expanded WOODW Problem

It was discovered empirically that selecting an initial dual subproblem with less than 6958 constraints resulted in an unbounded linear program. This problem was overcome computationally by slightly altering the LPC Algorithm 2.3.1 as follows. Instead of setting $[\bar{H}^j \quad \bar{b}^j]$ to be the set of active constraints at iteration j, the index set of constraints that are carried over to iteration $j+1$ consists of the union of the index set of all active constraints along which the objective is unbounded and the index set of positive dual variables. Specifically, consider a WOODW dual unbounded subproblem at iteration j. The subproblem constraint set is

$$\left\{ (u,v) \;\middle|\; \begin{array}{c} (A')^j u - (B')^j v \leq c^j, \\ (\bar{A}')^j u - (\bar{B}')^j v \leq \bar{c}^j, \\ v \geq 0 \end{array} \right\}. \tag{47}$$

The constraints then forming $[\bar{A}'^{j+1} \quad -\bar{B}'^{j+1} \quad \bar{c}^{j+1}]$ correspond to those all active constraints of (47) along which the objective $-b'u + d'v$ is unbounded below and the constraints of (47) with positive multipliers.

Unlike the implementation for data discrimination problems given in Section 2.3.3 which was done in C, this general linear programming implementation was done in MATLAB [110]. As such the capabilities of both the full linear program and the LPC methods of solution are restricted in terms of timing and memory management as a consequence of the MATLAB implementation. Hence the primary purpose of this section is to show the viability of the LPC Algorithm 2.3.1 for solving general linear programs.

Preliminary computational tests were carried out on the Computer Sciences Department Ironsides cluster of 4 Sun Enterprise E6000 machines, each with 16 UltraSPARC II processors for a total of 64 processors and 8 gigabytes of RAM using the MATLAB numerical package [110]. Linear programs were solved via the MINOS package [120] linked to MATLAB.

We first attempted computational tests in MATLAB [110] on a 98 megabyte Sun Sparcstation, but MATLAB memory problems were encountered. Hence we began testing on the 8 gigabyte Ironsides cluster using MATLAB. Given the large amount of memory available on the Ironsides cluster, the average time needed to solve the full WOODW primal problem (45) was merely 29.33 seconds. The short time needed to solve the full linear program motivated us to augmented the problem by tripling the number of primal variables or, equivalently, tripling the number of dual constraints. After the number of dual constraints was tripled, the expanded set of constraints were randomly reordered. We term this larger problem the "expanded WOODW" problem and the dual consists of 25215 constraints and 1098 variables.

The full expanded WOODW problem data was stored in MATLAB matrix-vector format as a binary .mat file [110]. To load the expanded WOODW problem data into the MATLAB environment, the process required 222 megabytes of resident machine memory. Different load times were observed, possibly due to different computational loads on the Ironsides cluster. Load times varied from tens of seconds to minutes. These observations should be kept in mind when interpreting the results summarized in Figure 10.

In solving the massive linear program via MINOS [120] called from within MATLAB, the MATLAB process required 1.069 gigabytes of resident memory from the Ironsides cluster. These figures suggest that the expensive costs incurred to compute solutions to large linear programs in the MATLAB environment. It is likely that computational resources required, in terms of time and memory, would be diminished if the MATLAB environment was avoided.

Average time to solve this augmented WOODW dual problem was 263 seconds, including the time needed to read the augmented WOODW problem data from disk. Data load time was included since, in the implementation of the LPC algorithm, the "new" subproblem data (that *not* carried over from a previous iteration) is read from disk.

Figure 10 summarizes time needed to obtain an optimal solution to the expanded

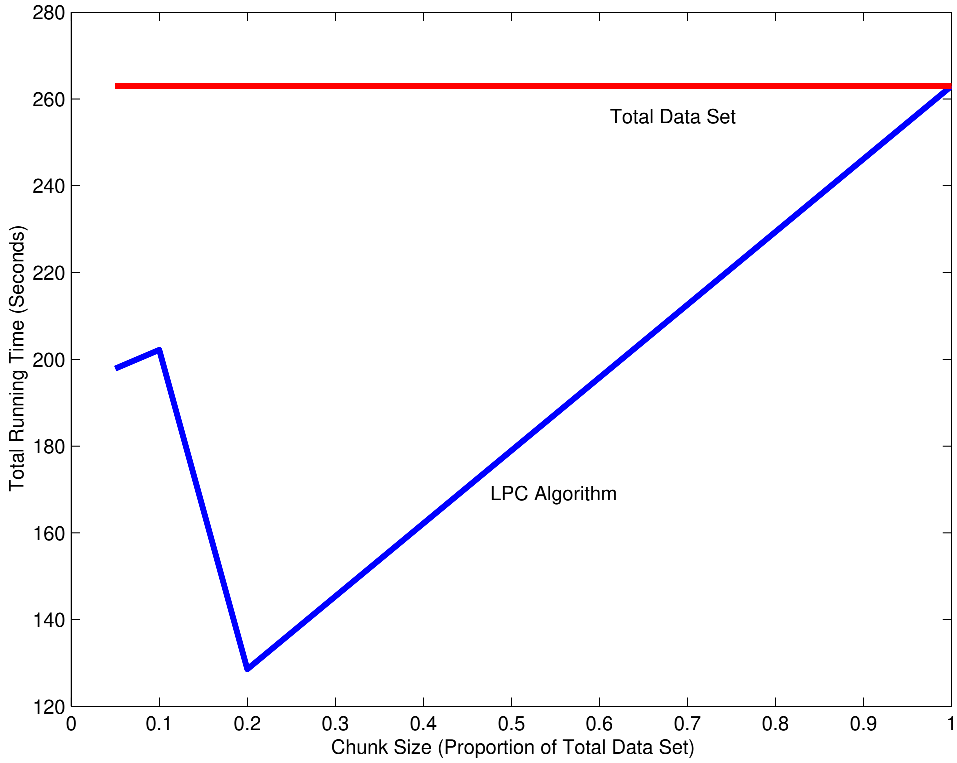


Figure 10: Total running time (seconds) versus chunk size for the LPC Algorithm 2.3.1 for the expanded WOODW dual problem on the CS Department Ironsides cluster (64 processors, 8 GB RAM).

WOODW dual problem for various initial subproblem sizes as a fraction of the total number of constraints of the expanded WOODW dual problem. Note that at each iteration, the actual subproblem solved consists of the given set of “new” data (equaling the given fraction of the total number of constraints) plus constraints carried over from the previous LPC iteration, as discussed above.

Note that all subproblem times are less than the time needed to load and solve the full expanded WOODW problem via MINOS [120] called from MATLAB [110]. The fact that there is not a noticeable trend in LPC times for various subproblem sizes could be due to the number of iterations required to construct a bounded subproblem.

This number varied over the different subproblem sizes tested. It was observed that for subproblem sizes requiring many iterations to reach a bounded subproblem, time to reach an optimal solution was also greater.

These results indicate the utility of the LPC algorithm for solving general linear programs with massive numbers of constraints.

2.3.3 LPC for Solving SVM-1

We consider applying the LPC Algorithm 2.3.1 to the SVM-1 problem (35) of calculating a separating plane $P := \{x \in R^n \mid w'x = \gamma\}$ to classify massive datasets

For clarity, we restate the SVM-1 problem:

$$\text{(SVM-1)} \min_{w,\gamma,y,z,s} \left\{ (1-\lambda)(e'y + e'z) + \frac{\lambda}{2}e's \;\middle|\; \begin{array}{c} -Aw + e\gamma + e \leq y, \\ Bw - e\gamma + e \leq z, \\ y \geq 0, z \geq 0, \\ -s \leq w \leq s. \end{array} \right\}, \quad \lambda \in (0,1).$$

For datasets $\mathcal{A}$ and $\mathcal{B}$ in R^n having m and k points respectively, the number of constraints in the SVM-1 problem (not including simple non-negativity constraints) is $m + k + 2n$. The number of variables is $2n + 1 + m + k$.

Consider the dual of the SVM-1 linear program (35):

$$\max_{u\in R^m, v\in R^k, p\in R^n, q\in R^n} \left\{ e'u + e'v \;\middle|\; \begin{array}{c} A'u - B'v + p - q = 0, \\ -e'u + e'v = 0, \\ u \le (1-\lambda)e, \\ v \le (1-\lambda)e, \\ p + q \le \frac{\lambda}{2}e, \\ u, v, p, q \ge 0 \end{array} \right\},$$

$$\lambda \in [0,1). \quad (48)$$

The number of constraints in this dual formulation (not including simple upper and lower bounds on the dual variables) is $2n+1$. The number of variables is $m+k+2n$. Solutions of the dual of SVM-1 (48) were faster to compute via MINOS [120] than solutions of the primal SVM-1 problem (35). Hence, when applying the LPC Algorithm 2.3.1 to the SVM-1 problem, all subproblems were solved in their dual form (48).

We note that the Sequential Minimal Optimization (SMO) technique [131] effectively scales the quadratic SVM-2 problem (36) to classify massive datasets. Each iteration consists of analytically computing the solution of 2 Lagrange multipliers corresponding to 2 training examples. Note that in both the LPC Algorithm 2.3.1 and the SMO algorithm, multiple data scans may be needed to find an optimal separating plane.

To test the LPC Algorithm on the classification task, synthetic data sets were generated using the publicly available Synthetic Classification Data Set Generator (SCDS)[113]. Fully dense data sets $\mathcal{A}$ and $\mathcal{B}$ of various sizes were created with 32 attributes. Points of $\mathcal{A}$ were generated according to the following rule: $(x_7 = 6) \wedge (x_{17} =$

3) $\wedge$ $(x_{19} = 10) \wedge (x_{29} = 6)$. Points of $\mathcal{B}$ were generated according to the following rule: $(x_{12} = 4) \wedge (x_{25} = 6) \wedge (x_{27} = 10) \wedge (x_{31} = 10)$. Then 10% of the data generated for $\mathcal{A}$ was labeled $\mathcal{B}$ and, similarly, 10% of the data generated for $\mathcal{B}$ was labeled $\mathcal{A}$ to simulate noise in labeling.

We used a small version of this dataset with 20,000 points for prototyping and obtaining accurate running times on a Sun SparcStation 20 with 98 megabytes of RAM. For larger problems with 200,000 points and more we used the Computer Sciences Department Ironsides cluster (see Section 2.3.2). All linear programs were solved with MINOS [120] called from a C implementation of the LPC algorithm. The parameter λ of the SVM-1 problem (35) was set to 0.05 in all runs.

Since the datasets generated by the SCDS generator [113] were constructed by decision rules which are rather complex to recover by a linear separator, we also evaluated the LPC Algorithm 2.3.1 applied to the SVM-1 problem (35) on the following sets $\mathcal{A}$ and $\mathcal{B}$ [115]. Data was sample from two multivariate Gaussian distributions in R^{32}. The first Gaussian had center $\mu^1 = (2.5)e$ and the second Gaussian had center $\mu^2 = 0$. Both Gaussians had covariance matrices equal to the 32×32 identity matrix. The set $\mathcal{A}$ consists of 9900 data points sampled from Gaussian 1 and 100 points sampled from Gaussian 2. The set $\mathcal{B}$ consists of 100 data points sampled from Gaussian 1 and 9900 sampled from Gaussian 2.

The SVM-1 problem (35) with $\lambda = 0.05$ computes a separating plane $w'x = \gamma$ such that 100 data points from $\mathcal{A}$ are "misclassified" or in the halfspace $\{x \in R^{32} \mid w'x < \gamma\}$ and 440 data points of $\mathcal{A}$ are in the region $\{x \in R^{32} \mid \gamma \leq w'x \leq \gamma + 1\}$. Hence there are 540 support vectors resulting from $\mathcal{A}$. Similarly, there are 100 data points of $\mathcal{B}$ which are "misclassified" in the halfspace $\{x \in R^{32} \mid w'x > \gamma\}$ and 437 points of $\mathcal{B}$ in

$\{x \in R^{32} \mid \gamma - 1 \leq w'x \leq \gamma\}$. Hence there are 537 support vectors from $\mathcal{B}$.

LPC Numerical Results for Massive Data Discrimination

Figures 11 and 12 show results for 200,000 points in R^{32} generated via SCDS [113] $(m + k = 200,000)$. Figure 11 depicts monotonic values of the minima of subproblems (38) at each iteration of the LPC algorithm converging to the global optimum of (37). Figure 12 depicts total running time of the LPC algorithm until a constant objective value is obtained. Note that the LPC algorithm reduced running time to 1.75 hours, a 74.8% reduction from the 6.94 hours for a full dataset linear program, when it used 12.5% of the data at a time. In fact for all chunk sizes less than 100% the LPC reduced running time.

Table 8 lists the maximum subproblem sizes in terms of number of data points considered and the size of the resulting dual subproblem constraint matrix. All LPC solutions were optimal for the full SVM-1 problem (35) over 200,000 data points.

Table 8: Maximum subproblem sizes for LPC applied to the SVM-1 (35) for discriminating between 200,000 points in R^{32}.

Original Chunk Size (% of full dataset)	Maximum # of Subproblem Constraints	Maximum Subproblem Dual Constraint Matrix Size
5%	$19,240$	$64 \times 19,304$
10%	$22,000$	$64 \times 22,064$
12.5%	$27,500$	$64 \times 27,564$
25%	$55,000$	$64 \times 55,064$
50%	$110,000$	$64 \times 110,064$

For the problem with 500,000 points in R^{32} generated via SCDS [113], the LPC algorithm with 12.5% chunking of the full dataset obtained a separating plane in 25.91

hours in 8 iterations. The objective function remained constant for the next $\nu = 4$ iterations. Note $m + k = 500,000$ and the maximum subproblem solved consisted of $62,500$ data points and the resulting dual constraint matrix was $65 \times 62,564$. The solution computed via the LPC algorithm was optimal for the SVM-1 problem (35) over the 500,000 data points. We were not able to obtain a solution for the full dataset linear program.

For the problem with 1 million points in R^{32} generated via SCDS [113], the LPC algorithm with 2% chunking of the full dataset obtained a separating plane in 231.32 hours in 63 iterations. The objective function remained constant for the next $\nu = 4$ iterations. It was not possible to obtain a solution for the full dataset linear program of this size. Note that $m+k = 1,000,000$ and the maximum subproblem solved consisted of $72,000$ data points and the resulting dual constraint matrix was $65 \times 72,064$.

For the datasets generated by sampling from the two Gaussian distributions discussed earlier, recall that the number of support vectors determined by the SVM-1 problem (35) over the full dataset is $540 + 537 = 1077$. We solved this problem with the initial number of subproblem constraints set to 1100 and $\lambda = 0.05$. The LPC Algorithm 2.3.1 (C implementation) computed an optimal solution in 788 seconds. In contrast the full linear program was solved via MINOS (called from C) in 2627 seconds. These tests were run on a 98 megabyte Sun Sparcstation.

These results indicate that if the number of support vectors can be estimated prior to the application of the LPC Algorithm 2.3.1, it may be easier to estimate the "correct" maximum number of constraints or chunk-size for any given subproblem [115], resulting in much quicker computation of an optimal solution.

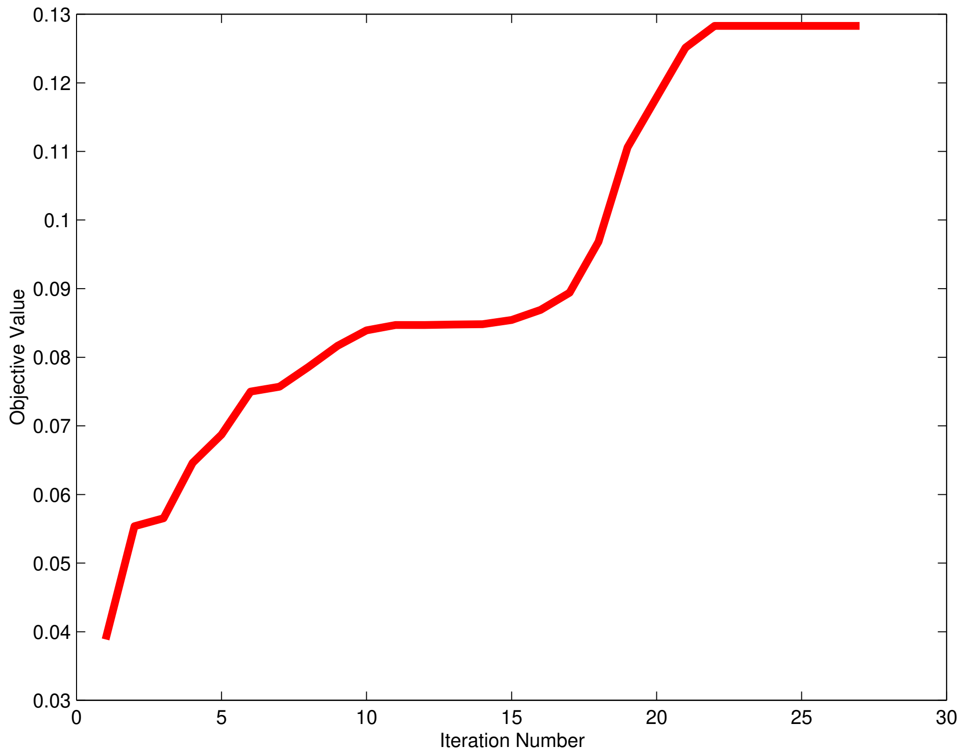


Figure 11: Objective function values for SVM-1 (35) versus LPC iteration number for chunk size of 5% of original 200,000 points in R^{32}. Objective function increases until iteration 22 and is flat thereafter.

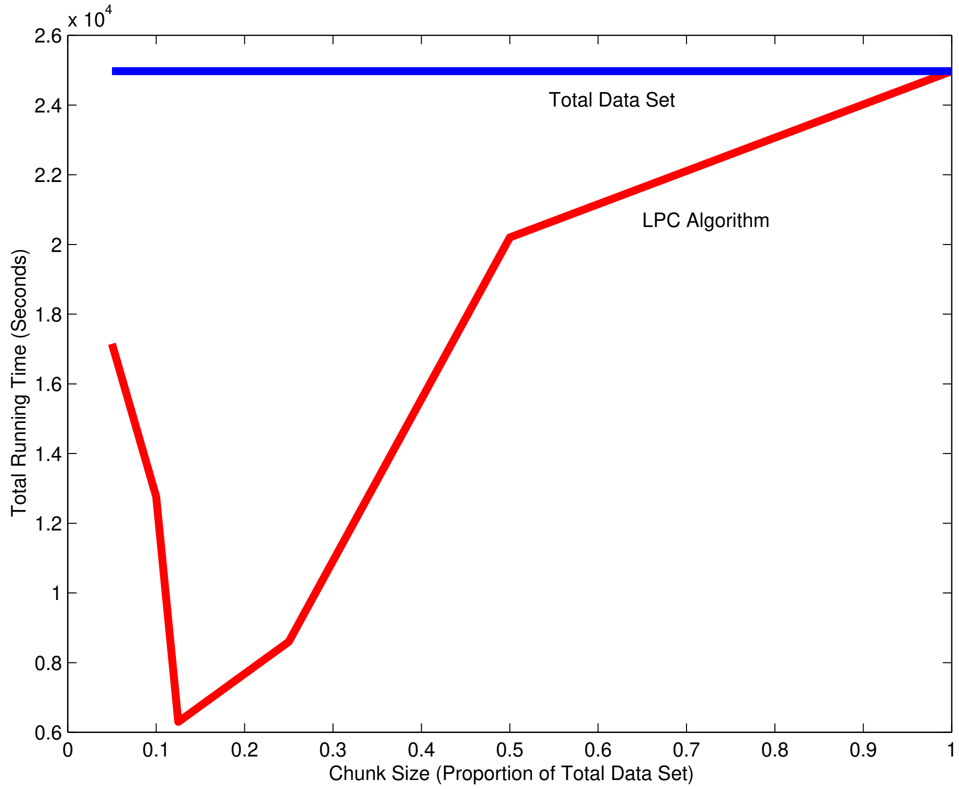


Figure 12: Total running time (seconds) versus chunk size for iterative LPC Algorithm 2.3.1 applied to SVM-1 (35) for dataset of 200,000 points in R^{32}. LPC time reduction of 74.8% at 12.5% chunk size.

The LPC Algorithm 2.3.1 presented in this section enables one to solve linear programs with massive constraints when limited by resident machine memory (RAM). This approach can be applied to any of the linear-programming-based classification approaches, enabling these to be solved over massive datasets often arising in real-world data mining tasks. In particular, the LPC algorithm addresses the data mining algorithm constraints presented in Section 1.3.1 in the following sense:

- *On-line "anytime" behavior:* a solution of the most recent iteration is always available from the LPC algorithm. Application of the LPC algorithm to the SVM-1 problem (35), yields a "best" separating plane at each iteration in the sense that it has been computed over the support vectors at each iteration. It may not be optimal over all of the data processed so far.

- *Ability to incrementally incorporate additional data* is addressed by viewing the additional data as a set (or sets) of constraints that need to be processed.

- *Work within the confines of a limited resident memory (RAM) buffer.* By partitioning the original constraint matrix correctly, the resulting subproblems (38) can be quickly and efficiently dealt with in main memory.

- *Require one scan of the dataset if possible.* Unfortunately, we cannot guarantee an optimal solution to the full massive linear program after a single scan of the dataset. In our computations we have observed an optimal solution after one to two full data scans when the LPC Algorithm 2.3.1 is applied to the SVM-1 problem (35).

In this chapter, the feature selection problem is cast as that of computing plane to separate the two disjoint point sets $\mathcal{A}$ and $\mathcal{B}$ utilizing as few of the original problem features as possible. The FSS (12), FSV (13) and FSB (16) problems were formulated. The Successive Linearization Algorithm 2.1.3 was used to solve the FSV problem, the minimization of a concave objective function over a polyhedral region. The support vector machine approach to classification was investigated and three formulations; SVM-∞ (34), SVM-1 (35) and SVM-2 (36) were compared with the FSV approach in addition to classifiers computed by the RLP (7). Computational results indicated that the FSV feature selection approach computes a separating plane with improved generalization ability over RLP planes. In comparison with the support vector machine approaches, the FSV planes defined simpler (in terms of fewer features utilized) classification functions with comparable generalization ability. The LPC Algorithm 2.3.1 was presented. This algorithm effectively solves linear programs with massive constraints. The LPC Algorithm scales the linear-programming-based methods to address massive classification tasks.

We next turn our attention to the problem of unsupervised clustering.

Chapter 3

Clustering via Mathematical Programming

We consider optimization approaches addressing the unsupervised clustering task (Section 1.2). The problem is that of assigning m points in R^n, represented by the data set $\mathcal{A} := \{x^i\}_{i=1}^m$, into k clusters. The clustering problem is first formulated as minimizing a piecewise linear concave function over a polyhedral set resulting in the k-Median algorithm. The k-Mean algorithm [147, 81, 63] computes a local solution to the problem of minimizing a nonconvex objective over a polyhedral set. We examine a novel approach, k-Plane clustering, where clusters are characterized by planes in R^n, obtainable by solving an eigenvalue problem for each cluster.

The algorithms presented above are shown to be effective data mining tools over moderately sized datasets, but have serious computational drawbacks when applied to massive datasets. In [18, 59], an algorithm is proposed for refining a given clustering initialization targeted at clustering massive datasets. In [19, 20], a scalable clustering framework is presented and instantiated on the k-Mean and EM algorithms. A scalable clustering approach is also presented in [168].

We next consider methods in which the clusters themselves are described by a centroid in R^n.

3.1 Clustering to Centroids

We consider the unsupervised assignment of elements of a given set to groups or clusters of like points. Many approaches to this problem include statistical [81, 63], machine learning [60], integer and mathematical programming [147, 3, 135].

We address the following explicit description of the clustering problem: given m points in n-dimensional real space R^n, and a fixed integer k of clusters, determine k centers in R^n such that the sum of the "distances" of each point to the nearest cluster center is minimized. If the 1-norm metric is used, the problem can be formulated as minimizing a piecewise-linear concave function over a polyhedral set. This is a difficult problem since a local minimum is not necessarily a global minimum. By converting this problem to an equivalent bilinear program, a fast successive-linearization k-Median algorithm terminates after a few iterations at a point satisfying a minimum principle necessary optimality condition [98]. Although there is no guarantee that such a point is a global solution, numerical tests reveal that the k-Median Algorithm 3.1.3 computes useful clustering solutions in diverse domains. The k-Median algorithm may be preferable in domains in which sensitivity to outliers is not desired, since the 1-norm is more robust in this case. In contrast the k-Mean algorithm uses squares of 2-norm distances to generate cluster centers, which may be sensitive to outliers. We note that clustering algorithms based on statistical assumptions that minimize some function of scatter matrices do not appear to have convergence proofs [63, pp. 508-515]. However convergence of a class of k-means-type algorithms to local solutions is given in [147].

We now consider formulating the clustering problem a bilinear program.

3.1.1 Clustering as Bilinear Programming

Given a set $\mathcal{A}$ of m points in R^n represented by the matrix $A \in R^{m\times n}$ and an integer k of desired clusters, we formulate the clustering problem as follows. Find k cluster centers C_ℓ, $\ell = 1, \ldots, k$, in R^n such that the sum of the minima over $\ell \in \{1, \ldots, k\}$ of the 1-norm distance between each point A_i, $i = 1, \ldots, m$, and the cluster centers $C_\ell, \ell = 1, \ldots, k$, is minimized. Specifically, consider the following mathematical program:

$$\min_{C,D} \left\{ \sum_{i=1}^{m} \min_{\ell=1,\ldots,k} (e'D_{i\ell}) \;\middle|\; \begin{array}{c} -D_{i\ell} \leq A_i' - C_\ell \leq D_{i\ell}, \\ i = 1, \ldots, m,\ \ell = 1, \ldots, k. \end{array} \right\}. \tag{49}$$

Here $D_{i\ell} \in R^n$ is a dummy variable bounding the components of the vector $|A_i' - C_\ell|$, the absolute component-wise difference between the data point A_i and the center C_ℓ. Hence $e'D_{i\ell}$ bounds the 1-norm distance between A_i and center C_ℓ. Note that since the objective function of (49) is the sum of the minima of k linear (and hence concave) functions, it is a piecewise-linear concave function [98, Corollary 4.1.14]. If the 2-norm or p-norm, $p \neq 1, \infty$, is used, the objective function will be neither concave nor convex. Nevertheless, minimizing a piecewise-linear concave function on a polyhedral set is NP-hard, because the general linear complementarity problem, which is NP-complete [43], can be reduced to such a problem [99, Lemma 1]. Given this fact, we focus our attention on effective methods for processing (49).

We propose reformulating problem (49) as a bilinear program. Similar reformulations have been very effective in computationally solving NP-complete linear complementarity problems [102] as well as other difficult machine learning [101] and optimization problems with equilibrium constraints [101]. In order to carry out this reformulation we need the following simple lemma.

Lemma 3.1.1 *Let $a \in R^k$. Then*

$$\min_{1 \leqq \ell \leqq k} \{a_\ell\} = \min_{t \in R^k} \left\{ \sum_{\ell=1}^{k} a_\ell t_\ell \,\middle|\, \sum_{\ell=1}^{k} t_\ell = 1,\ t_\ell \geq 0,\ \ell = 1, \ldots, k \right\} \tag{50}$$

Proof This essentially obvious result follows immediately upon writing the dual of the linear program appearing on the right-hand side of (50) which is

$$\max_{h \in R} \{h \,|\, h \leq a_\ell,\ \ell = 1, \ldots k\} \tag{51}$$

Obviously, the maximum of this dual problem is $h = \min_{1 \leqq \ell \leqq k} \{a_\ell\}$. By linear programming duality theory, this maximum equals the minimum of the primal linear program in the right hand side of (50). This establishes the equality of (50). □

By defining $a_\ell^i := e' D_{i\ell}$, $i = 1, \ldots, m$, $\ell = 1, \ldots, k$, Lemma 3.1.1 can be used to reformulate the clustering problem (49) as a bilinear program as follows.

Proposition 3.1.2 Clustering as a Bilinear Program *The clustering problem (49) is equivalent to the following bilinear program:*

$$\min_{C_\ell \in R^n, D_{i\ell} \in R^n, T_{i\ell} \in R} \left\{ \sum_{i=1}^{m} \sum_{\ell=1}^{k} T_{i\ell} \cdot (e' D_{i\ell}) \,\middle|\, \begin{array}{c} -D_{i\ell} \leq A_i' - C_\ell \leq D_{i\ell}, \\ i = 1, \ldots, m,\ \ell = 1, \ldots, k, \\ \sum_{\ell=1}^{k} T_{i\ell} = 1,\ i = 1, \ldots, m, \\ T_{i\ell} \geq 0, i = 1, \ldots, m,\ \ell = 1, \ldots, k. \end{array} \right\}. \tag{52}$$

Notice that the constraints of (52) are uncoupled in the variables (C, D) and the variable T. Hence the Uncoupled Bilinear Program Algorithm UBPA [9, Algorithm 2.1] is applicable. Simply stated, this algorithm alternates between solving a linear program in the variable T and a linear program in the variables (C, D). The algorithm terminates finitely at a stationary point satisfying the minimum principle necessary optimality condition for problem (52) [9, Theorem 2.1]. Further, because of the simple structure of the bilinear program (52), the two linear programs can be solved explicitly in closed form. This leads us to the following algorithmic implementation.

Algorithm 3.1.3 k-Median Algorithm *Given $C_1^j, \ldots, C_k^j$ at iteration j, compute $C_1^{j+1}, \ldots, C_k^{j+1}$ by the following two steps:*

(a) **Cluster Assignment:** *For each A_i', $i = 1, \ldots m$, determine $\ell(i)$ such that $C_{\ell(i)}^j$ is closest to A_i' in the 1-norm.*

(b) **Cluster Center Update:** *For $\ell = 1, \ldots, k$ choose C_ℓ^{j+1} as a median of all A_i' assigned to C_ℓ^j.*

Stop when $C_\ell^{j+1} = C_\ell^j$, $\ell = 1, \ldots, k$.

Since the problem (49) is the minimization of a piecewise-linear concave function over a polyhedral region, a successive linearization algorithm [107, Algorithm 1] is also applicable. This algorithm requires supergradients of the objective which are available in (49) and the resulting procedure reduces to Algorithm 3.1.3.

We next discuss the k-Mean approach to clustering [147, 81, 63] .

3.1.2 k-Mean Algorithm

The k-Mean algorithm is a standard technique for clustering. This procedure computes a local solution [147] to the following optimization problem.

$$\min_{C,D} \left\{ \sum_{i=1}^{m} \min_{\ell=1,\ldots,k} \frac{1}{2} \|D_{i\ell}\|_2^2 \;\middle|\; \begin{array}{c} -D_{i\ell} \le A_i' - C_\ell \le D_{i\ell}, \\ i = 1, \ldots, m, \; \ell = 1, \ldots, k \end{array} \right\}. \tag{53}$$

The k-Mean algorithm differs from the k-Median Algorithm 3.1.3 in the the cluster assignment step: a point is assigned to the cluster with center nearest in the 2-norm. The cluster update step consists of computing the mean of the data assigned to the given cluster, instead of the median.

Algorithm 3.1.4 k-Mean Algorithm *Given* $C_1^j, \ldots, C_k^j$ *at iteration* j*, compute* $C_1^{j+1}, \ldots, C_k^{j+1}$ *by the following two steps:*

(a) Cluster Assignment: *For each* A_i'*,* $i = 1, \ldots m$*, determine* $\ell(i)$ *such that* $C_{\ell(i)}^j$ *is closest to* A_i' *in the 2-norm.*

(b) Cluster Center Update: *For* $\ell = 1, \ldots, k$ *choose* C_ℓ^{j+1} *as the mean of all* A_i' *assigned to* C_ℓ^j*.*

Stop when $C_\ell^{j+1} = C_\ell^j$*,* $\ell = 1, \ldots, k$*.*

The k-Median and k-Mean algorithms differ not only computationally but also theoretically. In fact, the underlying problem (49) of the k-Median algorithm is a concave minimization over a polyhedral set, while the corresponding problem for the p-norm, $p \neq 1, \infty$ is:

$$\min_{C,D} \left\{ \sum_{i=1}^{m} \min_{\ell=1,\ldots,k} \|D_{i\ell}\|_p \;\middle|\; \begin{array}{c} -D_{i\ell} \leq A_i' - C_\ell \leq D_{i\ell}, \\ i = 1, \ldots, m,\ \ell = 1, \ldots, k. \end{array} \right\}. \tag{54}$$

This is not a concave minimization over a polyhedral set, because the minimum of a set of convex functions is not in general concave. The concave minimization problem of [147] is not in the original space of the problem variables, but merely in the space of the the variables T that assign points to clusters. We also note that the k-Mean Algorithm

3.1.4 finds a stationary point not of problem (54), but of the same problem with $\|D_{i\ell}\|_2$ replaced with $\|D_{i\ell}\|_2^2$ (53). Without the squared distance term, the subproblem of the k-Mean algorithm becomes the considerably harder Weber problem [128, 45] which locates a center in R^n closest in sum of Euclidean distances (not their squares) to a given finite set of points. The Weber problem has no closed form solution. However, using the mean as a center of points assigned to the cluster minimizes the sum of the *squares* of the distances from the cluster center to the points. It is precisely the mean that is used in the k-Mean algorithm subproblem.

We focus on evaluating the k-Median Algorithm 3.1.3 and the k-Mean Algorithm 3.1.4 in three clustering tasks.

3.1.3 Data Mining Survival Curves

In many medical domains, survival curves [84] are important prognostic tools. Our goal was use the k-Median Algorithm 3.1.3 and the k-Mean Algorithm 3.1.4 as data mining tools in a KDD process over two medical datasets to identify groups with distinct survival characteristics.

We used an altered version of the WPBC dataset [119]. For a description of the dataset as it exists at the UCI ML Repository, please see Section 2.1.4. The four instances which are missing the value for feature 35 were removed. We utilized only feature 34 and feature 35 corresponding to tumor size (diameter of the excised tumor in centimeters) and lymph node status (number of positive axillary lymph nodes observed at time of surgery). These two features were then normalized to have mean = 0 and standard deviation = 1. In this data-mining context, when referring to the WPBC dataset, we are referring to this set of 194 points in R^2.

We also used a subset of the SEER database [35] consisting of the two features tumor size and nodes positive for 21,960 instances. Each of these features is encoded in a non-intuitive fashion as an integer value in $\{0, \ldots, 8\}$. Tumor size = 0 corresponds to “no tumor found or microscopic tumor only”. Tumor size = 1 corresponds to tumor size less than 0.5 cm. Tumor size = 2 corresponds to tumor size between 0.5 cm and 0.9 cm. Tumor size = 3 corresponds to tumor size between 1.0 cm and 1.9 cm. Tumor size = 4 corresponds to size between 2.0 cm and 2.9 cm. Tumor size = 5 corresponds to size between 3.0 cm and 3.9 cm. Tumor size = 6 corresponds to 4.0 cm to 4.9 cm. Tumor size = 7 corresponds to size between 5.0 cm and 9.9 cm. Tumor size = 8 corresponds to a tumor of size 10 cm or larger. A value of nodes positive between 0 and 7 corresponds to actual number of positive axillary lymph nodes observed at time of surgery. Nodes positive = 8 corresponds to 8 or more positive nodes. When referring to the SEER dataset, we are referring to this set of 21,960 points in R^2.

We applied the k-Median and k-Mean algorithms with $k = 3$, as data mining tools, to the WPBC and SEER datasets. Survival curves [84] were then constructed for each cluster, representing expected percent of surviving patients as a function of time, for patients in that cluster. The value of $k = 3$ was chosen in the hope of determining clusters that represented patients with “good”, “average” and “poor” prognosis, as depicted by the survival curves.

Clustering each of these datasets on the Computer Sciences Department Ironsides Cluster (see Section 2.3.2) requires time on the order of seconds. Hence a “repetition with random restarts” strategy was employed [55] to address the problem of local optimality (recall both the k-Median and k-Mean solutions are dependent upon the initial cluster centers). Sets of initial centers were generated by sampling 20 sets of $k =$ 3 cluster centers from a uniform distribution on the range of the data. Figure 13 depicts the "best" survival curve obtained over 20 random initializations for three groups computed by the k-Median Algorithm 3.1.3. Figure 14 depicts the "best" survival curves over the same 20 random initializations computed by the k-Mean Algorithm 3.1.4. "Best" curves were chosen by visually analyzing the separability of the survival curves computed for each cluster. Evaluating whether the survival characteristics are distinct for each cluster could also be performed by comparing p-values for pairwise log rank statistics computed from each survival curve [86].

The survival curves of clusters determined by the k-Median Algorithm 3.1.3 (Figure 13) are more separated than those determined by the k-Mean Algorithm 3.1.4 (Figure 14). These results indicate the utility of these clustering approaches as data mining tools.

To further evaluate these algorithms in a medical survival domain, the k-Median Algorithm 3.1.3 and the k-Mean Algorithm 3.1.4 were applied to the SEER dataset. Clusters were obtained from 20 sets of $k = 3$ initializations sampled uniformly on the range of the SEER dataset. Figure 15 depicts survival curves for three clusters obtained by the k-Median Algorithm 3.1.3 on the SEER dataset. Figure 16 depicts curves computed by the k-Mean Algorithm 3.1.4. The curves are not identical, but one can conclude that both methods identify clusters with distinct survival characteristics as determined by the survival curves.

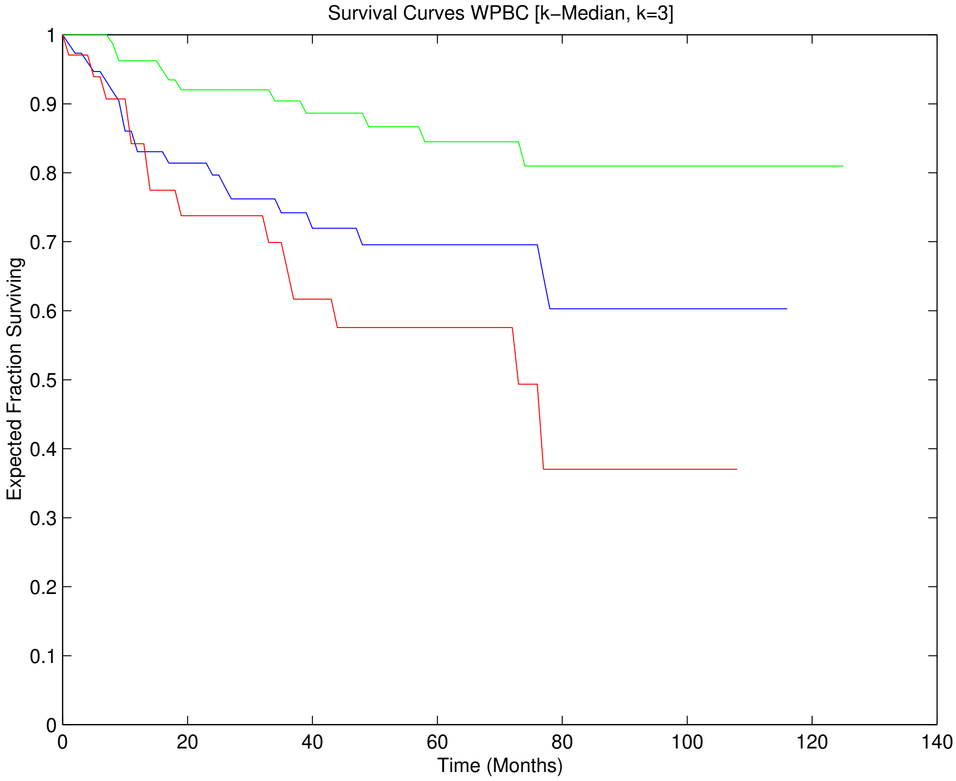


Figure 13: Estimated fraction of disease free patients versus time (months) for 3 clusters obtained with the k-Median Algorithm 3.1.3 on the WPBC dataset.

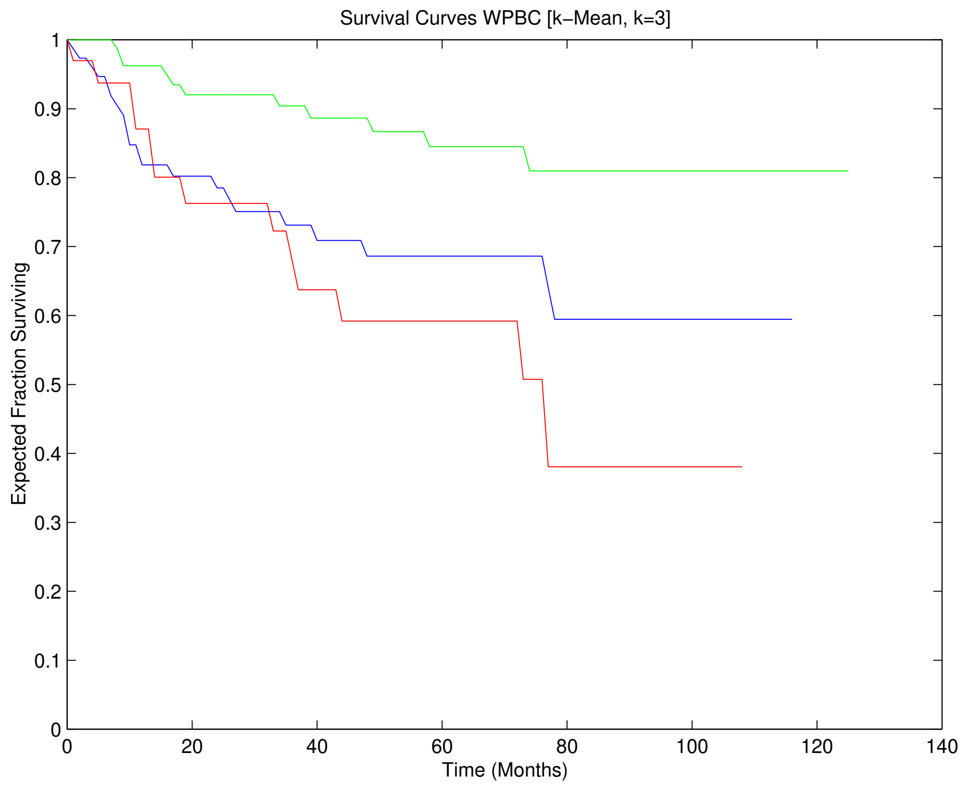


Figure 14: Estimated fraction of disease free patients versus time (months) for 3 clusters obtained with the k-Mean Algorithm on the WPBC dataset.

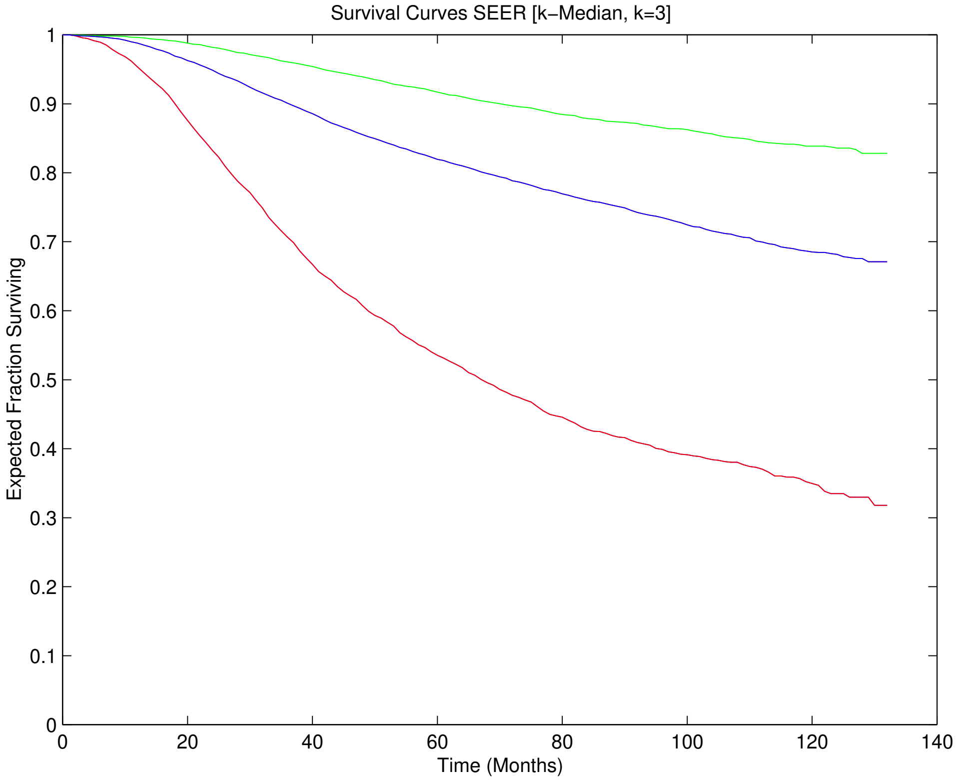


Figure 15: Estimated fraction of disease free patients versus time (months) for 3 clusters obtained with the k-Median Algorithm 3.1.3 on the SEER dataset.

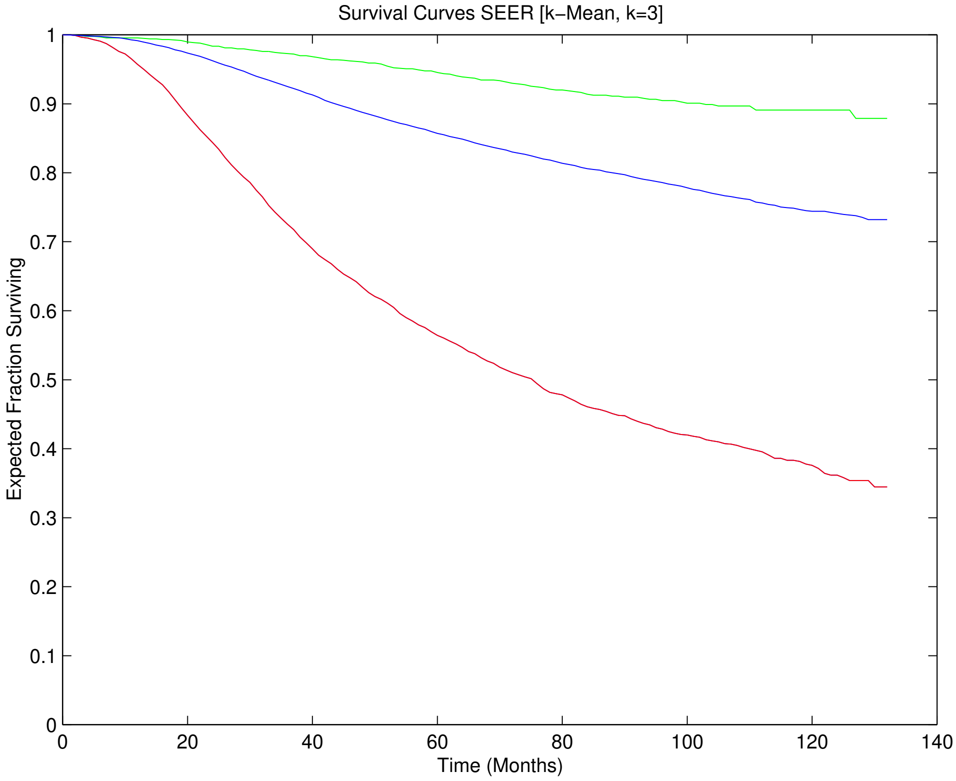


Figure 16: Estimated fraction of disease free patients versus time (months) for 3 clusters obtained with the k-Mean Algorithm 3.1.4 on the SEER dataset.

3.1.4 Training Set Correctness

We further evaluated the k-Median Algorithm 3.1.3 and the k-Mean Algorithm 3.1.4 by the following procedure. The algorithms are applied with $k = 2$ to a dataset with two known classes to obtain 2 centers. Training set correctness is measured by the ratio of of the sum of the number of examples in the majority class in each cluster to the total number of points in the dataset. Training correctness of the k-Median and k-Mean algorithms is compared to that of a 2-class classification estimator obtained by solving the RLP (7). Initial centers for the k-Median and k-Mean algorithms were deterministically chosen by dividing the coordinate axes into six intervals over the range of the data and choosing two centers as the midpoints of the densest and second densest intervals on the axes [27].

We utilized three publicly available databases from the UCI ML Repository [119] and one dataset that appears in [126].

The Wisconsin Diagnostic Breast Cancer Dataset (WDBC) consists of 569 instances each having 32 features. Feature 1 is an identification and was discarded. Feature 2 is diagnosis (M = malignant, B = benign) and was used to assign instances to the two datasets $\mathcal{A}$ and $\mathcal{B}$. Features 3-32 are the mean, standard error, and "worst" measurements of ten real-valued features computed by the **Xcyt** [154] program at time of diagnosis. When referring to the WDBC dataset, we refer to the point set $\mathcal{A} \subset R^{30}$ consisting of features 3-32 for 357 instances which have a benign diagnosis and the set $\mathcal{B} \subset R^{30}$ consisting of the same features for 212 instances with a malignant diagnosis. Each feature has been normalized to have mean = 0 and standard deviation = 1.

The Cleveland Heart Disease dataset consists of 303 instances with each instance having 14 features. Six instances have a missing feature value, these were discarded from the dataset. Feature 14 indicates the presence of heart disease and is used to assign data points to the two sets $\mathcal{A}$ and $\mathcal{B}$. For the semantics of features 1-13, please see [119]. When referring to the Cleveland dataset, we refer to the point set $\mathcal{A} \in R^{13}$ consisting of features 1-13 for 214 instances in which feature 14 has value 0 or 1 and the point set $\mathcal{B} \in R^{13}$ consisting of the same features for 83 instances in which feature 14 has value 2, 3, or 4. Each feature has been normalized to have mean = 0, standard deviation = 1.

The 1984 United States Congressional Voting Records database includes votes from each of the U.S. House of Representatives Congressmen on 16 key votes identified by the CQA [119]. There are 435 instances, each instance represents the voting record of a Representative ("yea", "nay", or "unknown disposition"). Set $\mathcal{A} \subset R^{16}$ consists of 267 instances representing the voting records for Democratic Representatives. Set $\mathcal{B} \subset R^{16}$ consists of 168 instances representing the voting records for Republican Representatives. Each feature has been normalized to have mean = 0, standard deviation = 1.

The Star/Galaxy-Bright dataset [126] consists of the set $\mathcal{A} \subset R^{14}$ having 1505 examples and the set $\mathcal{B} \subset R^{14}$ having 957 examples. Each of the 14 features has been normalized to have mean = 0, standard deviation = 1.

Results are summarized in Table 9. We note that for two of the databases the k-Median Algorithm 3.1.3 outperformed the k-Mean Algorithm 3.1.4, and for the other two k-Mean was better.

Table 9: Training Set Correctness

Algorithm ↓ Database →	WDBC	Cleveland	Votes	Star/Galaxy-Bright
Unsupervised k-Median	0.932	0.806	0.846	0.876
Unsupervised k-Mean	0.911	0.831	0.855	0.856
Supervised Robust LP	1.000	0.865	0.95.6	0.997

3.1.5 Testing Set Correctness

We further evaluated the k-Median Algorithm 3.1.3 and k-Mean Algorithm 3.1.4 in a situation motivated by data mining large databases. We assume that we have a large database of examples which we wish to classify. This can be thought of as a data mining step in the KDD process mentioned in Section 1.3. The difficulty is that only a small fraction of the available examples have been classified as being in one of two disjoint sets, by a domain expert for instance. We examine two possible solutions to classifying the large fraction currently unclassified and estimating correctness of this classification.

(i) Determine two clusters (with the k-Median or k-Mean algorithms) over the unclassified examples. Classify the individual examples by cluster membership. Estimate classification correctness using the class label of the classified examples.

(ii) Determine a separating plane (3) using the the classified examples, by solving the RLP (7). Classify individual examples by the classification function defined by the separating plane. Estimate classification correctness by a leave-one-out strategy [90] using the classified examples.

We simulate the situation in which only a fraction of a database has been classified by splitting the WDBC dataset into two subsets S_1 and S_2. We view the points of S_1 as being classified and the points of S_2 as without classification. The k-Median and k-Mean algorithms ($k = 2$) are applied to the set S_2 to obtain two cluster centers. Initial centers are chosen by dividing the coordinate axes into six intervals over the range of the data in S_2 and choosing two centers as midpoints of the densest and second densest intervals. We then assign the points in S_1 to clusters with nearest center (1-norm for k-Median, 2-norm for k-Mean). Clusters are given a class tag by majority membership (e.g. if cluster one contains mostly points of $\mathcal{B}$, then it is tagged "$\mathcal{B}$"). Testing set correctness is determined by the fraction of points in S_1 correctly classified by this assignment. The WDBC dataset used here is the same as described in Section 3.1.4.

Test set correctness for the estimated classification function determined by solving the RLP (7) is estimated by a leave-one-out strategy [90] on set S_1.

Figure 17 depicts results averaged over 50 runs for each of 7 sizes for set S_1 (the *testing subset*), as a percentage of the size of the entire WDBC dataset. As expected, the performance of the classification function determined by solving the robust linear program (7) improves as the size of S_1 increases. The k-Median Algorithm 3.1.3 test set correctness remained fairly constant in the range of 92.3% to 93.5%, while the k-Mean Algorithm 3.1.4 test set correctness was lower and more varied in the range 88.0% to 91.3%.

The k-Median Algorithm 3.1.3 is a useful data mining tool particularly suited to determining clusters which are "robust" to outliers since it is a 1-norm based algorithm. In contrast the k-Mean Algorithm 3.1.4 is a 2-norm *squared* based algorithm making

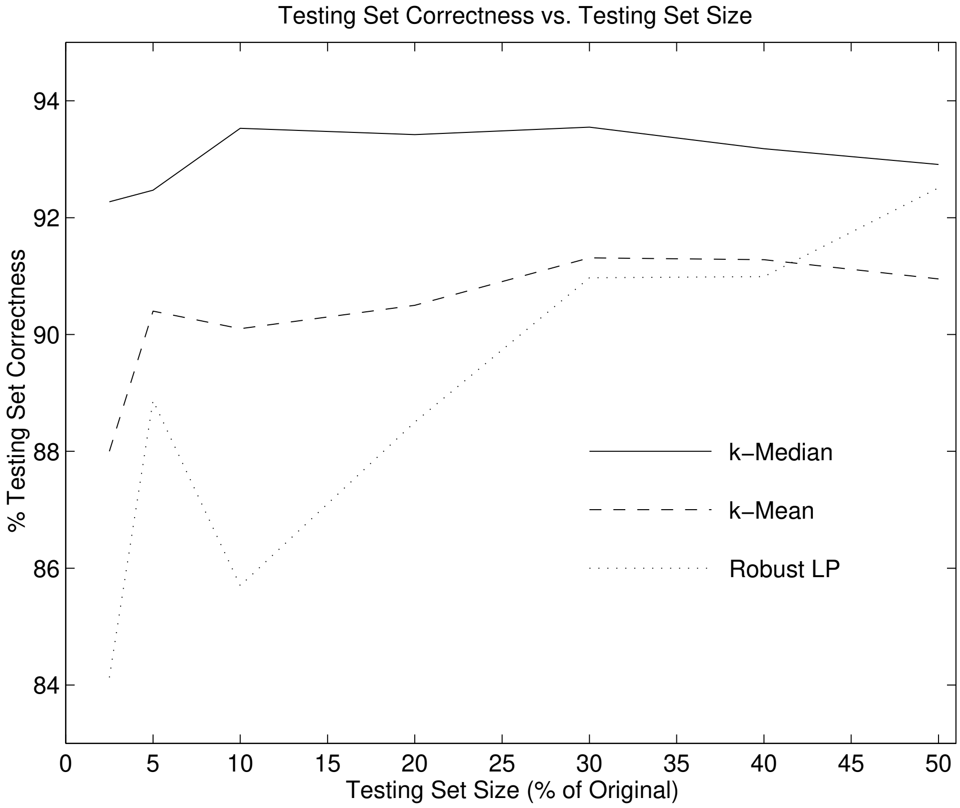


Figure 17: Correctness on variable-size test set of the unsupervised k-Median and k-Mean Algorithms versus correctness of the supervised robust linear program on the WDBC dataset.

it sensitive to outliers. The computational evaluations of this section indicated that both are viable data mining tools addressing clustering tasks in medical domains.

Note that the k-Median Algorithm 3.1.3 computes a solution which is locally optimal. We cannot guaranteed a solution which is globally optimal.

The issue of initializing iterative clustering approaches is discussed in [18, 59].

We now focus on a novel clustering approach in which the clusters are represented by planes in contrast to points in R^n [23].

3.2 k-Plane Clustering

We change the characterization of a cluster from a centroid or point in R^n to a plane in R^n. The justification for this approach is that data sometimes naturally falls into clusters grouped around flat surfaces such as planes. This approach yields interesting theoretical results that lead to an efficiently implementable algorithm.

3.2.1 k-Plane Clustering Algorithm

We consider a set $\mathcal{A}$ of m points in n-dimensional real space R^n represented by the matrix $A \in R^{m\times n}$. We wish to cluster $\mathcal{A}$ into k clusters according to the following to the following rule. Determine k cluster planes in R^n:

$$P_\ell := \{x \,|\, x \in R^n, x'w_\ell = \gamma_\ell\}, \quad \ell = 1, \ldots, k, \tag{55}$$

that minimize the sum of the squares of distances of each point of $\mathcal{A}$ to its nearest plane P_ℓ. The algorithm is similar to the k-Median Algorithm 3.1.3 in that it alternates between assigning points to a nearest cluster plane and, for a given cluster, computing a cluster plane that minimizes the sum of the squares of distances to all points in the cluster. It is the latter computation, which is a one step replacement of an algorithm for the Euclidean Regression Problem [150, 37] which does not use squared distances and cannot be solved in one step, that makes the following k-Plane clustering algorithm possible.

Algorithm 3.2.1 **k-Plane Clustering Algorithm** *Start with initial planes* (w_1^0, γ_1^0), $\ldots$, (w_k^0, γ_k^0), *each in* R^{n+1} *with* $\|w_i^0\|_2 = 1$, $i = 1, \ldots, k$. *Having* $(w_1^j, \gamma_1^j), \ldots, (w_k^j, \gamma_k^j)$ *at iteration* j *with* $\|w_i^j\|_2 = 1$, $i = 1, \ldots, k$, *compute* $(w_1^{j+1}, \gamma_1^{j+1}), \ldots, (w_k^{j+1}, \gamma_k^{j+1})$ *by the following two steps:*

(a) **Cluster Assignment:** *(Assign each point to closest plane* P_ℓ*) For each* A_i, $i = 1, \ldots m$, *determine* $\ell(i)$ *such that*

$$|A_i w_{\ell(i)}^j - \gamma_{\ell(i)}^j| = \min_{\ell=1,\ldots,k} |A_i w_\ell^j - \gamma_\ell^j|.$$

(b) **Cluster Update:** *(Find a plane* P_ℓ *that minimizes the sum of the squares of distances to each point in cluster* ℓ*) For* $\ell = 1, \ldots, k$ *let* $A(\ell)$ *be the* $m(\ell) \times n$ *matrix with rows corresponding to all* A_i *assigned to cluster* ℓ. *Define*

$$B(\ell) := [A(\ell)]'(I - \frac{ee'}{m(\ell)})A(\ell).$$

Set w_ℓ^{j+1} *to be an eigenvector of* $B(\ell)$ *corresponding to the smallest eigenvalue of* $B(\ell)$. *Set* $\gamma_\ell^{j+1} := \dfrac{e'A(\ell)w_\ell^{j+1}}{m(\ell)}$.

Stop whenever $(w_\ell^{j+1}, \gamma_\ell^{j+1}) = (w_\ell^i, \gamma_\ell^i)$, $\ell = 1, \ldots, k$, *for some* $i = j, j-1, \ldots, 0$.

We give in the next section the theoretical justification for the k-Plane Algorithm 3.2.1 and establish its finite termination.

3.2.2 Theoretical Justification of the k-Plane Algorithm

We first note that the cluster assignment rule defined in Step (a) of the k-Plane Algorithm 3.2.1 follows from the well known fact [104] that the 2-norm distance between between a point $A_i \in R^n$ and the plane $P_\ell := \{x \,|\, x \in R^n, x'w_\ell = \gamma_\ell\}$ is $|A_i w_\ell - \gamma_\ell|/\|w_\ell\|_2 = |A_i w_\ell - \gamma_\ell|$. The last equality follows from $\|w_\ell\|_2 = 1$.

The cluster update rule defined in Step (b) of the k-Plane Algorithm 3.2.1 follows from Theorem 3.2.6 below. But first we prove a few simple lemmas.

Lemma 3.2.2 *Let $A \in R^{m\times n}$. Then,*

$$\left\langle \begin{array}{c} Aw - e\gamma = 0,\ w \neq 0 \\ \text{has no solution } (w,\gamma) \end{array} \right\rangle \Leftrightarrow \left\langle \begin{array}{c} rank(A) = n, \text{ and} \\ Aw = e \text{ has no solution } w \end{array} \right\rangle. \tag{56}$$

Proof ($\Rightarrow$) If $rank(A) < n$, then $Aw - e \cdot 0 = 0$, $w \neq 0$ has a solution which is a contradiction. If $Aw = e$ has a solution, then $Aw - e(1) = 0$, $w \neq 0$ has a solution which is again a contradiction.

($\Leftarrow$) If $Aw - e\gamma = 0$, $w \neq 0$ has a solution, then either $\gamma = 0$ or $\gamma \neq 0$. In the first case, $rank(A) < n$. In the second case, by dividing by γ, we have that $Aw = e$ has a solution. In either case, a contradiction ensues. □

Lemma 3.2.3 *Let $A \in R^{m\times n}$, then*

$$A'(I - \frac{ee'}{m})A = A'(I - \frac{ee'}{m})^2 A. \tag{57}$$

Proof

$$(I - \frac{ee'}{m})^2 - (I - \frac{ee'}{m}) = I - 2\frac{ee'}{m} + \frac{ee'ee'}{m^2} - I + \frac{ee'}{m} = 0. \ \square \tag{58}$$

Lemma 3.2.4 *$B := A'(I - \frac{ee'}{m})A$ is positive semidefinite.*

Proof By Lemma 3.2.3,

$$w'Bw = \|(I - \frac{ee'}{m})Aw\|_2^2 \geq 0. \ \square \tag{59}$$

Lemma 3.2.5 *$Aw - e\gamma = 0$, $w \neq 0$ has no solution $\Leftrightarrow$ B is positive definite.*

Proof ($\Rightarrow$) By Lemma 3.2.4, B is positive semidefinite. If B is *not* positive definite then, by Lemma 3.2.4, $(I - \frac{ee'}{m})Aw = 0$, $w \neq 0$ has a solution. But, by Lemma 3.2.2, $rank(A) = n$, hence $z = Aw \neq 0$. Thus, $(I - \frac{ee'}{m})z = 0$, or $z = e\frac{e'z}{m} = \alpha e$, where $\alpha = \frac{e'z}{m}$. Since $z \neq 0$, it follows that $\alpha \neq 0$ and $e = \frac{z}{\alpha} = A\frac{w}{\alpha}$, contradicting the fact (from Lemma 3.2.2) that $Aw = e$ has no solution.

($\Leftarrow$) If B is positive definite, then by Lemma 3.2.4, $(I - \frac{ee'}{m})Aw = 0$ has no solution $w \neq 0$. Hence $\mathrm{rank}(A) = n$. Also $Aw = e$ has no solution, else $(I - \frac{ee'}{m})Aw = e - e = 0$. Thus by Lemma 3.2.2, $Aw - e\gamma = 0$, $w \neq 0$, has no solution. $\square$

We are ready now to state the theorem that explicitly gives the plane that minimizes the sum of the squares of the 2-norm distances to m given points in R^n.

Theorem 3.2.6 *Let $A \in R^{m\times n}$. Then a global solution of:*

$$\begin{array}{ll} \underset{(w,\gamma)\in R^{n+1}}{\text{minimize}} & \|Aw - e\gamma\|_2^2 \\ \text{subject to} & w'w = 1, \end{array} \tag{60}$$

is attained at any eigenvector w of $B := A'(I - \frac{ee'}{m})A$ corresponding to a minimum eigenvalue of B and $\gamma = \frac{e'Aw}{m}$. The minimum of (60) is positive if and only if B is positive definite or equivalently if and only if $rank(A) = n$ and $Aw = e$ has no solution.

Proof The second part follows from Lemmas 3.2.2 and 3.2.5. We now prove the first part. The set of all stationary points of (60) including all its global minima render the partial derivatives of the Lagrangian of (60) equal to zero. That is for:

$$L(w,\gamma,\lambda) := \|Aw - e\gamma\|_2^2 - \lambda(w'w - 1), \tag{61}$$

it follows that:

$$\frac{1}{2}\nabla_w L(w,\gamma,\lambda) = A'(Aw - e\gamma) - \lambda w = 0, \tag{62}$$

$$-\frac{1}{2}\nabla_\gamma L(w,\gamma,\lambda) = e'(Aw - e\gamma) = 0, \tag{63}$$

$$-\nabla_\lambda L(w,\gamma,\lambda) = w'w - 1 = 0. \tag{64}$$

Hence:

$$\lambda = w'A'(I - \frac{ee'}{m})Aw, \tag{65}$$

$$\gamma = \frac{e'Aw}{m}. \tag{66}$$

Substitution for λ and γ in (62) gives:

$$A'(I - \frac{ee'}{m})Aw - w'A'(I - \frac{ee'}{m})Aw \cdot w = 0. \tag{67}$$

By using the definition of B this is equivalent to:

$$Bw - w'Bw \cdot w = 0.$$

That is:

$$Bw = \nu w, \ \nu = w'Bw. \tag{68}$$

Thus for each stationary point (w, γ) of (60), it follows that w is an eigenvector of B and $\gamma = \frac{e'Aw}{m}$. Hence,

$$Aw - e\gamma = Aw - e(\frac{e'Aw}{m}) = (I - \frac{ee'}{m})Aw. \tag{69}$$

We then have by Lemma 3.2.3 that:

$$\|Aw - e\gamma\|_2^2 = w'A'(I - \frac{ee'}{m})^2 Aw = w'Bw = \nu, \tag{70}$$

where the last equality follows from (68). Hence the smallest value that ν can take on is the smallest eigenvalue of B and w is its corresponding eigenvector. □

Remark 3.2.7 Relation to Singular Value Decomposition *It can be shown, after some straightforward algebra, that the w obtained in the above Theorem 3.2.6 can also be obtained by taking a singular value decomposition USV' [125, 153] of the $m \times n$ matrix:*

$$H := (I - \frac{ee'}{m})A,$$

where U and V are orthogonal matrices of dimensions $m \times m$ and $n \times n$ respectively, and S is an $m \times n$ diagonal matrix with nonnegative diagonal elements in decreasing order. It can then be shown that the desired w given by Theorem 3.2.6 corresponds to the last column of the matrix V corresponding to a smallest singular value of H, and γ is again given by (66) above. This result can be derived by noting that [125, Theorem 8.19] the squares of the singular values of H (possibly with some zeros added) are also the eigenvalues of both HH' and $H'H$ with associated eigenvectors being columns of U

and V respectively. A different clustering approach, latent semantic indexing, is given in [10] that also uses singular value decomposition.

We end this section by establishing the finiteness of the k-Plane Algorithm 3.2.1.

Theorem 3.2.8 (Finite Termination of the k-Plane Algorithm 3.2.1) *The k-Plane Algorithm 3.2.1 terminates in a finite number of steps at a cluster assignment that is locally optimal. That is, the overall objective, the sum of the squares of distances of each point to a closest cluster plane, cannot be decreased by either reassignment of a point to a different cluster plane, or by defining a new cluster plane for any of the clusters.*

Proof In the cluster assignment part (a) of the algorithm each point is assigned to a closest plane and hence the overall objective cannot increase. Similarly in part (b) of the algorithm, the cluster plane, for each cluster, is recomputed as that plane which minimizes the sum of the squares of distances of points in that cluster to the plane. Hence, again, the overall objective cannot increase. Since there is finite number of ways that the m points of $\mathcal{A}$ can be assigned to k clusters, since the algorithm does not permit repeated assignments, and since the overall objective function is non-increasing and bounded below by zero, it follows that the algorithm must terminate at some clustering assignment that is locally optimal. □

3.2.3 Computational Results

Two sets of computational tests were carried out to evaluate the k-Plane Clustering Algorithm 3.2.1 The first test is designed to investigate the ability to generate well separated survival curves by clustering medical data (see Section 3.1.3). In the second set of tests the ability to recover class labels by clustering unlabeled data was tested.

The first set of tests consisted of applying the k-Plane algorithm to a subset of the WPBC dataset [119] with goal of obtaining separated survival curves [84]. The two features of tumor size and lymph node status were used (see Section 3.1.3). Both features were normalized to have zero mean and standard deviation = 1. This dataset consists of 198 points in R^2. The k-Plane algorithm was run from 20 sets of $k = 3$ initial planes obtained by sampling the values of (w_1^0, γ_1^0), (w_2^0, γ_2^0), (w_3^0, γ_3^0) from a normal distribution with mean = 0, standard deviation = 1. The k-Plane Algorithm 3.2.1 was applied to with $k = 3$ and survival curves corresponding to each cluster were plotted. The "best" set of survival curves (determined visually) are depicted in Figure 19 indicating three groups with distinct survival characteristics.

The actual clusters obtained in R^2 are depicted in Figure 18.

Application of the k-Plane Clustering Algorithm to the SEER Database (see Section 3.1.3) also yielded 3 clusters with distince prognostic characteristics. See Figure 20.

In the second set of tests the k-Plane algorithm, k-Median and the k-Mean algorithm were compared in their ability to recover class labels on a holdout data subset. The datasets used here had two classes, hence $k = 2$ in these tests. A ten-fold cross-validation [152] scheme was employed. In this procedure the dataset is randomly divided into 10 disjoint sets of approximately equal size, $T_1, T_2, \ldots, T_{10}$. Then 10 trials are conducted. At trial j, the clustering algorithms are applied to the union of $T_1, \ldots, T_{j-1}, T_{j+1}, \ldots, T_{10}$ (training data) without making use of the class label. Then the data points in T_j (test data) were assigned to the closest cluster plane or cluster center. Training correctness at trial j is the percentage of training data

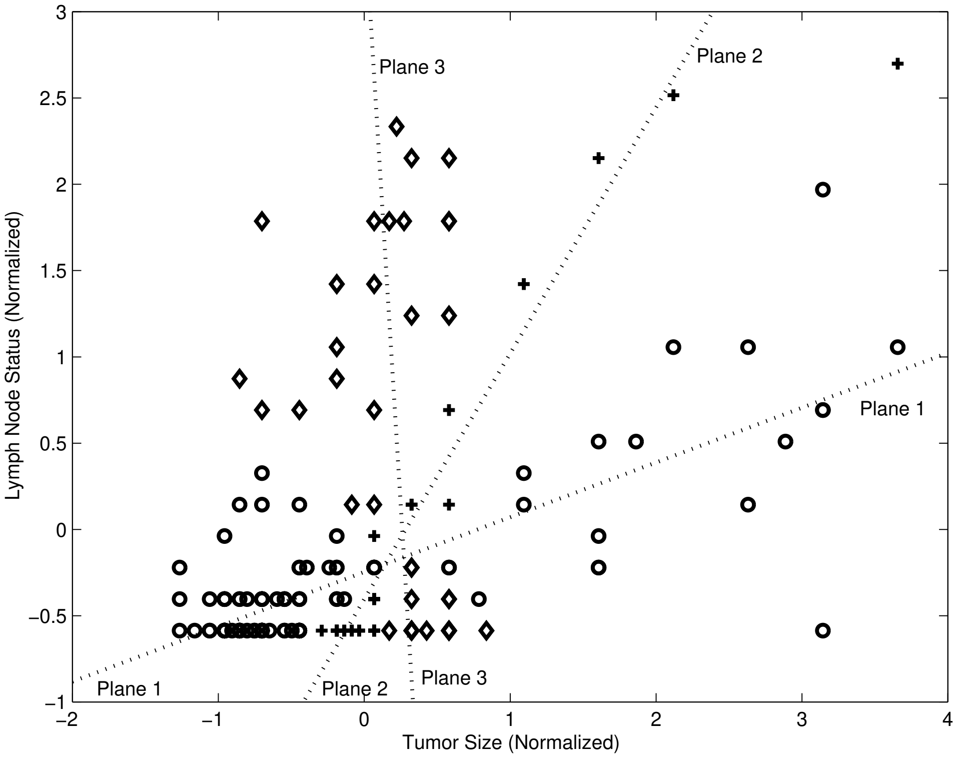


Figure 18: Three cluster lines obtained by the k-Plane Algorithm 3.2.1 for the Wisconsin Prognostic Breast Cancer Database (WPBC). Data assigned to Plane 1 is indicated by $\bigcirc$. Data assigned to Plane 2 is indicated by $+$. Data assigned to Plane 3 is indicated by $\diamond$.

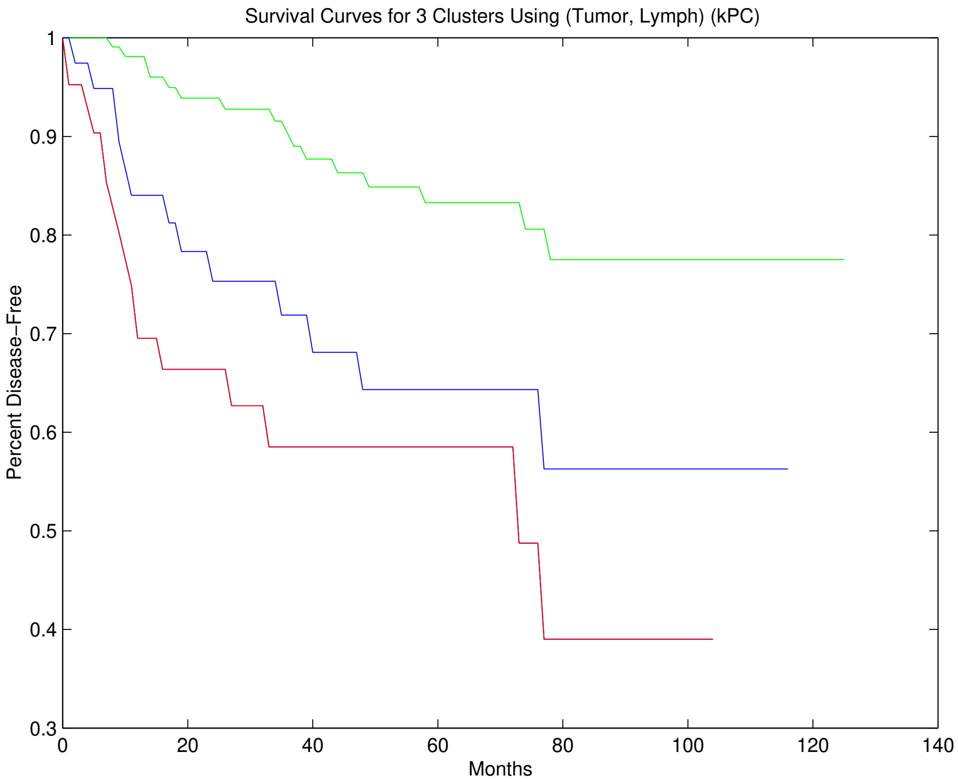


Figure 19: Estimated fraction of disease free patients versus time (months) for 3 clusters obtained with the k-Plane Clustering Algorithm 3.2.1 on the WPBC dataset.

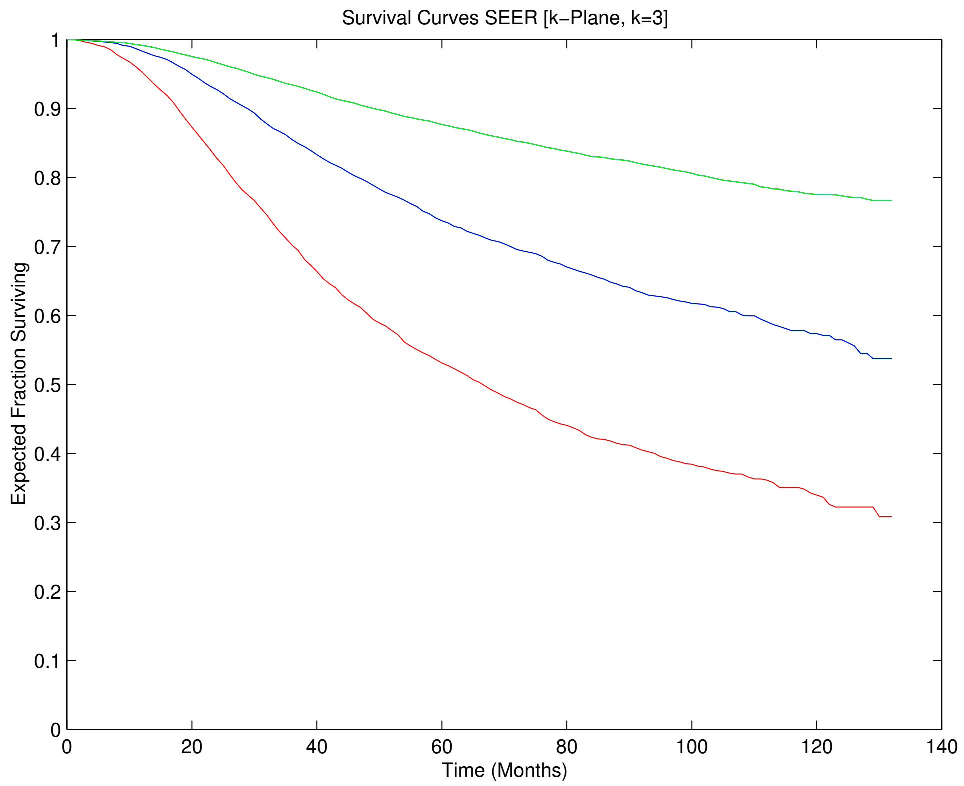


Figure 20: Estimated fraction of disease free patients versus time (months) for 3 clusters obtained with the k-Plane Clustering Algorithm 3.2.1 on the WPBC dataset.

$T_1, T_2, \ldots, T_{j-1}, T_{j+1}, \ldots, T_{10}$ correctly classified by the majority label of the cluster that each point was assigned to. Similarly, testing correctness at trial j is the percentage of T_j correctly classified by the majority label of the cluster that the point was assigned to.

Table 10 summarizes average training and testing results on 2 publicly available datasets [119] for the k-Plane, k-Mean and k-Median algorithms.

The Johns Hopkins Ionosphere dataset consists of 351 data points with 34 real-valued features characterizing radar returns from the ionosphere. One class corresponds to radar returns showing evidence of structure. The other class corresponds to those returns showing no structure.

The BUPA Liver Disorders dataset consists of 345 data points with 6 real-valued features. A 7th feature indicates the class of the corresponding feature.

Both the Ionosphere and BUPA datasets have been normalized so that the mean of each feature is zero and standard deviation is one. Initial planes for the k-Plane Algorithm 3.2.1 and initial centers for the k-Median Algorithm 3.1.3 and the k-Mean Algorithm 3.1.4 where sampled from a normal distribution with zero mean and standard deviation 1.

We note that the k-Plane clusters were better able to recover original class labels on the BUPA liver disorders dataset over both the training and testing subsets. The k-Mean clusters were better on the Ionosphere dataset. On both the BUPA and Ionosphere datasets, the k-Plane algorithm converged faster than k-Mean, as much as 6.21 times faster on the BUPA dataset [23].

The novel k-Plane Algorithm 3.2.1 provides the analyst with a clustering tool which

Table 10: 10-fold Cross-Validation Results

		Ionosphere	BUPA
Ave. Test Correct.	k-Plane	0.641 $\pm$ 0.011	0.603 $\pm$ 0.047
	k-Mean	0.706 $\pm$ 0.100	0.556 $\pm$ 0.032
	k-Median	0.661 $\pm$ 0.109	0.565 $\pm$ 0.033
Ave. Train Correct.	k-Plane	0.641 $\pm$ 0.001	0.555 $\pm$ 0.028
	k-Mean	0.704 $\pm$ 0.015	0.549 $\pm$ 0.009
	k-Median	0.659 $\pm$ 0.012	0.519 $\pm$ 0.009

Value following "$\pm$" is one standard deviation.

may be very useful when inherent groupings in the data are based upon a subspace effectively approximated by a plane. This efficient, theoretically justifiable technique computes a cluster assignment which is locally optimal. Computational results indicated its utility in a medical survival data mining application and in a class-label recovery task.

We now turn our attention to a more general problem of obtaining a minimum support solution [106] to a linear system $Ax = b$.

Chapter 4

Parsimonious Approximation

A wide range of important applications, including those in machine learning [163, 41], can be reduced to the problem of estimating a vector $x \in R^n$ by minimizing some norm of the residual vector $Ax - b$ arising from a system of linear equations:

$$Ax = b, \tag{71}$$

where $A \in R^{m \times n}$ and $b \in R^m$, both A and b are subject to error.

We discuss other approaches to this problem before discussing a novel method, based on minimizing a concave function over a polyhedral set.

4.1 Other Approaches

There are various approaches to the problem of estimating the vector $x \in R^n$ such that $Ax \approx b$. These include least squares, total least squares and structured total least norm.

The least squares (LS) solution, x_{LS} to (71) is obtained by solving the following problem [80]:

$$\min_{x \in R^n} \|Ax - b\|_2. \tag{72}$$

Suppose that the RHS b is corrupted by uncorrelated random variables with zero mean and equal variance, then the estimate x_{LS} has the smallest variance in the class of estimation methods which fulfill the following two conditions: (1) the estimate is unbiased, and (2) the estimate is a linear function of b [80].

The total least squares (TLS) problem [72, 80] (or orthogonal regression, or errors-in-variables regression) addresses the following optimization problem:

$$\min_{[\hat{A};\hat{b}] \in R^{m\times(n+1)}} \left\{ \|[A;b] - [\hat{A};\hat{b}]\|_F \;\middle|\; \hat{b} \in R(\hat{A}) \right\}, \tag{73}$$

where $\|\cdot\|_F$ is the Frobenius norm of a matrix and $R(\hat{A})$ is the column space of the matrix $\hat{A}$. Once a minimizing $[\hat{A};\hat{b}]$ is found, then $x_{\mathrm{TLS}} \in R^n$ satisfying

$$\hat{A} x_{\mathrm{TLS}} = \hat{b} \tag{74}$$

is called the total least squares solution. Notice that the noisy data $[A;b]$ are minimally modified (in the Frobenius norm) to a "nearby" system $[\hat{A};\hat{b}]$ which is solvable.

If $[A;b] = [A_0 + \triangle A, b_0 + \triangle b]$, where $[\triangle A, \triangle b]$ are uncorrelated random variables with zero mean and equal variance, it can be shown [64, 70] that x_{TLS} estimates $x_0 := A_0^\dagger b_0$ consistently in the sense that x_{TLS} converges to x_0 as m tends to infinity. Here $A_0^\dagger$ is the pseudo-inverse of A_0 obtained from the singular value decomposition of A_0 [80].

We formulate the total least squares problem as in [138] which leads us to a description of the structured total least norm problem. The TLS problem can be formulated as determining a perturbation matrix E with minimal norm while minimizing the residual $r = b - (A+E)x$:

$$\min \|E; r\|_F \tag{75}$$

In many applications the matrix A has special structure, such as Toeplitz or a sparsity structure. In addition, in some applications, errors may only occur in certain elements of A, in which case E is sparse. But the computational method most often employed to solve the TLS problem is based on the singular value decomposition (SVD) [73], which is not in general sparsity preserving or structure preserving. In contrast structured total least norm (STLN) methods [138, 79] permit a known structure in A and $[A; b]$ to be preserved in $A+E$ and $[A+E; b+r]$. Furthermore, the problem can be formulated so as to minimize the 1-norm, ∞-norm or the Frobenius norm used in the TLS problem. In [138], iterative methods are presented for solving the STLN problem for a given norm, convergence results and optimality conditions are investigated, and a comparison of STLN with LS and TLS is presented.

4.2 Parsimonious Least Norm Approximation

In this section we consider the closely related problem of minimizing the 1-norm of the residual vector $Ax - b$, where b is subject to error and with the additional condition that a specified number $k < n$ of the columns of A are used. This is clearly a combinatorial problem that we shall solve by minimizing a concave function on polyhedral set. This approach has been successfully used in such machine learning problems as misclassification minimization [101], feature selection [27] and data mining [26, 105].

The idea behind using as few columns of A as possible to span b is motivated by the Occam's Razor bias [13]. This bias is effective in improving the generalization ability of an estimator [148, 167] where, for example, one wishes to use the solution x of (71) on new data not represented by the rows of $[A\ b]$, as would be the case if either A or b is corrupted by noise.

Utilizing as few columns of A as possible is equivalent to computing a minimal support solution [106] of (71). To obtain a minimum support solution to (71), the objective function consist of weighting the 1-norm of the residual vector by $(1-\lambda)$ and weighting by λ a term which suppresses the components of x, which is our Parsimonious Least Norm Approximation (PLNA) problem:

$$\min_{x \in R^n} (1-\lambda)\|Ax-b\|_1 + \lambda e'|x|_*, \quad \lambda \in [0,1). \tag{76}$$

For comparative purposes we shall also employ Vapnik's support vector machine approach [161, 6] of minimizing the size of the solution vector x as well as the error $\|Ax-b\|_1$, thereby decreasing the VC-Dimension [161, p 76] (a capacity measure) and improving generalization. We shall do that by parametrically minimizing the 1-norm of x as well as the 1-norm of the error $Ax-b$:

$$\min_{x \in R^n} (1-\lambda)\|Ax-b\|_1 + \lambda\|x\|_1, \quad \lambda \in [0,1). \tag{77}$$

We shall call this problem, with a possibly noise-corrupted b, the *least* least norm approximation (LLNA) problem and solve it by solving the equivalent linear programming formulation:

$$\min_{(x,y,z)\in R^{n+m+n}} \left\{ (1-\lambda)e'y + \lambda e'z \;\middle|\; \begin{array}{c} -y \le Ax - b \le y, \\ -z \le x \le z \end{array} \right\}, \; \lambda \in [0,1). \tag{78}$$

Remark 4.2.1 *The following example where the 1-norm linear system residual may decrease but the number of nonzeros in the solution x increases in moving from vertex to vertex of (78) was suggested by R. R. Meyer [114]. Suppose the linear system (with perturbation vector $p = 0$) is:*

$$A = \begin{bmatrix} 1 & 0 \\ 2 & 1 \\ -1 & 1 \end{bmatrix}, b = \begin{bmatrix} 1 \\ 3 \\ 4 \end{bmatrix}.$$

One vertex of (78) is $\tilde{x} = [0\ 3]'$ and $\|A\tilde{x} - b\|_1 = 2$. Another vertex of (78) is $\bar{x}' = [1\ 1]$ with $\|A\bar{x} - b\|_1 = 4$. Note that the number of nonzeros of $\tilde{x}$ is less than the number of nonzeros of $\bar{x}$.

Remark 4.2.2 *The issue of whether vertex solutions of problem (78) with few nonzero elements (but greater than zero) have smaller norms is due to R. R. Meyer [114], who has also provided the following result. There exists a vertex solution of (78) with one nonzero component of x with norm no bigger than the norm of the "full solution" x^* obtained when $\lambda = 0$ in (78), provided $x^* \neq 0$.*

We note that when $\lambda = 0$, problem (76) and (77) reduce to the classic 1-norm approximation problem. When $\lambda = 1$, both problems (76) and (77) are trivially solved by $x = 0$.

We are interested in solutions with $\lambda \in [0,1)$ that make $e'|x|_* \leqq k$ for some desired $k < n$ and such that $\|Ax - b\|_1$ is acceptably small. By letting λ range over the interval $[0,1]$, the number of nonzero elements in the solution x varies from n to 0, while the error $\|Ax - b\|_1$ monotonically increases. Depending on the problem at hand, a solution for a $\lambda \in [0,1)$ will be most desirable. For instance, in many machine learning applications, λ is normally chosen as the one that produces a solution with best generalization estimation, determined by the use of a tuning set, for instance.

In the following section, we convert (76) to a minimization of a concave function over a polyhedral set. We state some theoretical and computational aspects of this conversion.

4.3 The PLNA Concave Minimization Problem

We first rewrite (76) as the following equivalent problem

$$\min_{(x,y)\in R^{n+m}} \left\{(1-\lambda)e'y + \lambda e'|x|_* \;\middle|\; -y \leq Ax - b \leq y\right\},\ \lambda \in [0,1). \tag{79}$$

Then by making the following identifications,

$$\begin{aligned} S &= \{(x,y) \mid -y \leq Ax - b \leq y\}, \\ \ell &= n + m, \\ s &= [x'\ y']', \\ h &= [e'\ 0']', \\ f(s) &= e'y, \\ \mu &= \tfrac{\lambda}{1-\lambda}, \end{aligned} \tag{80}$$

problem (79) and hence problem (76) become special cases of the following problem,

$$\min_{x \in S} f(x) + \mu h'|s|_* \tag{81}$$

which we shall solve in its smooth version,

$$\min_{s \in S} f(x) + \mu h'(e - \varepsilon^{-\alpha|s|}) \tag{82}$$

where α is a positive parameter. More specifically, the smooth version of (76) is the following concave minimization problem:

$$\min_{(x,y,z) \in R^{n+m+n}} \left\{ (1-\lambda)e'y + \lambda e'(e - \varepsilon^{-\alpha z}) \;\middle|\; \begin{array}{c} -y \leq Ax - b \leq y, \\ -z \leq x \leq z \end{array} \right\}, \ \lambda \in [0,1). \tag{83}$$

By solving this problem for a sufficiently large but finite value of $\alpha > 0$ it follows by Theorem 2.1.2 that we have solved our original discontinuous problem (76). We now turn our attention to solving (83) by a finitely terminating successive linearization algorithm.

4.4 The Concave Minimization Algorithm

The finite method that we propose is the successive linear approximation (SLA) method of minimizing a concave function on a polyhedral set which is a finitely terminating stepless Frank-Wolfe algorithm [62], similar to Algorithm 2.1.3. We state now the SLA for problem (83) which has a differentiable concave objective.

Algorithm 4.4.1 Successive Linearization Algorithm (SLA) for (83). *Choose* $\lambda \in [0, 1)$. *Start with a random* $x^0 \in R^n$. *Set* $y^0 = |Ax^0 - b|$, $z^0 = |x^0|$. *Having* (x^i, y^i, z^i) *determine* $(x^{i+1}, y^{i+1}, z^{i+1})$ *by solving the following linear program:*

$$(x^{i+1}, y^{i+1}, z^{i+1}) \in \arg\min_{(x,y,z)\in R^{n+m+n}} \left\{ (1-\lambda)e'y + \lambda\alpha(\varepsilon^{-\alpha z^i})'z \;\middle|\; \begin{array}{c} -y \le Ax - b \le y, \\ -z \le x \le z \end{array} \right\}. \tag{84}$$

Stop when (x^i, y^i, z^i) *is feasible and*

$$(1-\lambda)e'y^i + \lambda\alpha(\varepsilon^{-\alpha z^i})'z^i = (1-\lambda)e'y^{i+1} + \lambda\alpha(\varepsilon^{-\alpha z^i})'z^{i+1}. \tag{85}$$

By [103, Theorem 4.2] we have the following finite termination result for the SLA (Algorithm 4.4.1).

Theorem 4.4.2 Finite Termination for SLA applied to (83). *The iterates determined by (84) generate a strictly decreasing sequence of objective function values for (83) and terminates at an iteration $\bar{i}$ with a stationary point (which may also be a global minimum solution) that satisfies the following minimum principle necessary optimality criterion [98],*

$$(1-\lambda)e'(y-y^{\bar{i}}) + \lambda\alpha(\varepsilon^{-\alpha z^{\bar{i}}})'(z-z^{\bar{i}}) \geq 0, \ \forall \textit{ feasible } (x,y,z). \tag{86}$$

We now turn our attention to numerical testing of Algorithm 4.4.1 and the linear programming formulation (78).

4.5 Application and Numerical Testing

We wish to determine whether x-component suppression or x-norm reduction of an observed linear system $Ax = b + p$ which is a corruption of a true system $Ax = b$, leads to an improved approximation of the true system. One can relate this to a machine learning framework by treating the observed system as a *training set*, and the true system as a *testing set* [75]. The linear systems used are based upon ideas related to signal processing [69, 155] and more specifically to an example in [2, Equation (8)].

We consider the following *true* signal $g(t) : [0,1] \longrightarrow R$:

$$g(t) = \sum_{j=1}^{3} x_j \varepsilon^{-a_j t}, \ t \in [0,1], \ a = [0 \ \ 4 \ \ 7]', \ x = [0.5 \ \ 2.0 \ \ -1.5]'. \tag{87}$$

We assume that the true signal $g(t)$ cannot be sampled precisely, but that the following *observed* signal can be sampled:

$$\tilde{g}(t) = (g(t) + \mathrm{error}),\ \ sampled\ \ at\ \ times:\ t_i = i \triangle t,\ \triangle t = 0.04,\ i = 0, 1, \ldots, 25. \tag{88}$$

We further assume that we do not know the true signal $g(t)$ (87), and we attempt to model it as:

$$\hat{g}(t) = \sum_{j=1}^{10} x_j \varepsilon^{-a_j t},\ t \in [0, 1],\ a = [0\ \ 4\ \ 7\ \ 0.1\ \ 2\ \ 3\ \ 3.9\ \ 4.1\ \ 6.9\ \ 7.1]'. \tag{89}$$

The parameter values a in (89) were chosen as in [2, Equation (8)].

The problem now is to compute the coefficients $x_j,\ j = 1, \ldots, 10$, of $\hat{g}(t)$ (89) so that we can adequately recover $g(t)$, given only the noisy data $\tilde{g}(t_i)$ (88). Notice that by substituting the following coefficient vector x^* into (89), $\hat{g}(t) = g(t)$:

$$x^* := [0.5\ \ 2.0\ \ \ -1.5\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0\ \ 0]'. \tag{90}$$

Thus the *true* linear system (testing set) $Ax = b$ is then given by:

$$A_{ij} = \varepsilon^{-a_j t_i},\ b_i = g(t_i),\ i = 0, \ldots, 25,\ j = 1, \ldots 10, \tag{91}$$

and is solved exactly by x^* of (90).

The *observed* linear system (training set) $Ax = b + p$ is then given by:

$$\left\langle \begin{array}{l} A_{ij} = \varepsilon^{-a_j t_i},\ \ b_i = g(t_i), \\ p_i = random\ number\ with\ mean\ = 0\ \&\ standard\ deviation\ = 1, \\ i = 0, \ldots, 25,\ j = 1, \ldots, 10. \end{array} \right\rangle \tag{92}$$

We will refer to a solution of problem (76), with b of (76) replaced by $b+p$, computed by the Algorithm 4.4.1 as a *PLNA solution.* Similarly, we shall refer to a solution of problem (77), with b replaced by $b+p$ as an *LLNA solution.* We note here that for *all* experiments, the value of α in the negative exponential of (76) is 5.0. Scalars are considered zero if they are in the interval $[-1e-8, 1e-8]$. The components of the initial starting point x^0 for Algorithm 4.4.1 were sampled from a normal distribution with mean = 0 and standard deviation = 1, then the initial point was *fixed* for all runs as:

$$x^0 = [-0.5077 \;\; 0.8853 \;\; -0.2481 \;\; -0.7262 \;\; -0.4450 \\ -0.6129 \;\; -0.2091 \;\; 0.5621 \;\; -1.0639 \;\; 0.3516]'. \quad (93)$$

We now focus our attention on four approaches and compare solutions obtained by the PLNA and LLNA methods with solutions obtained by least squares and by a combinatorial search.

4.5.1 Comparison of PLNA, LLNA and Least Squares

We compute solutions of the observed system $Ax = b+p$, where A, b, and p are defined in (92), by PLNA, LLNA and by least squares. These solutions are then evaluated by the observed system (training set) residual $\|Ax-b-p\|_1$ and the true system (testing set) residual $\|Ax-b\|_1$ and graphically comparing the recovered signal $\hat{g}(t)$ (89) to the true signal $g(t)$ (87).

The PLNA solution $x(\lambda)$ of $Ax = b+p$, for a given λ is computed by Algorithm 4.4.1 solving the concave minimization problem (83) with b replaced by $b+p$ as follows:

$$\min_{(x,y,z)\in R^{n+m+n}} \left\{ (1-\lambda)e'y + \lambda e'(e - \varepsilon^{-\alpha z}) \;\middle|\; \begin{array}{c} -y \leq Ax - b - p \leq y, \\ -z \leq x \leq z \end{array} \right\},\ \lambda \in [0,1). \tag{94}$$

The LLNA solution $x(\lambda)$ of $Ax = b + p$, for a given λ is computed by solving the linear program (78) with b replaced by $b + p$ as follows:

$$\min_{(x,y,z)\in R^{n+m+n}} \left\{ (1-\lambda)e'y + \lambda e'z \;\middle|\; \begin{array}{c} -y \leq Ax - b - p \leq y, \\ -z \leq x \leq z \end{array} \right\},\ \lambda \in [0,1). \tag{95}$$

The least squares solution is a minimizer of $\|Ax - b - p\|_2$ and is a solution to the normal equations:

$$A'Ax = A'(b + p). \tag{96}$$

Although the 26×10 matrix A defined by (92) has rank 10, the matrix $A'A$ is numerically singular with smallest eigenvalue less than 10^{-14}. Thus we resort to a singular value decomposition approach for solving (96).

We determine an approximate solution $x(ls)$ to (96) by the following method which utilizes the singular value decomposition [153]. Ordinary MATLAB [110] commands such as $x = A\backslash(b + p)$ for our perturbed system $Ax = b + p$ give an x with an error $\|x - x^*\|_2 = 2.1379e + 08$ compared to $\|x - x^*\|_2 = 2.6675e + 03$ given by the method described below, where x^* is defined by (90) and the perturbation vector p components are sampled from a normal distribution with mean $= 0$, standard deviation $= 1$.

Algorithm 4.5.1 Least Squares via Singular Value Decomposition. *Let $A \in R^{m\times n}$ with $m \geq n$. Let τ be a small positive tolerance.*

1. *Determine the economy singular value decomposition of A [110, svd(A,0)], $U \in R^{m\times n}$, $S \in R^{n\times n}, V \in R^{n\times n}$:*

$$A = USV', \tag{97}$$

where $U'U = V'V = I_n$ (the $n \times n$ identity matrix), and $S = diag(\sigma_1, \ldots, \sigma_n)$, $\sigma_i \geq \sigma_{i+1} \geq 0$, $i = 1, \ldots, n-1$.

2. *Determine the index r such that $\sigma_i \geq \tau$ for $i = 1, \ldots, r$.*

3. *Set $\tilde{U} \in R^{m\times r}$ to be the first r columns of U, $\tilde{V} \in R^{n\times r}$ to be the first r columns of V and $\tilde{S} \in R^{r\times r}$ to be $diag(\sigma_1, \ldots, \sigma_r)$.*

4. *Compute $x(ls) = \tilde{V}\tilde{S}^{-1}\tilde{U}'(b+p)$, which is a solution to:*

$$\min_{x\in\tilde{T}} \frac{1}{2}x'x, \quad \tilde{T} := \{x \mid \tilde{V}'x = \tilde{S}^{-1}\tilde{U}'(b+p)\ \} \approx \{x \mid A'Ax = A'(b+p)\}. \tag{98}$$

For all runs τ was fixed at 0.0001, which for our specific matrix A defined by (92), led to $r = 6$ in the above algorithm. That is we discarded the last 4 columns of U and V.

The PLNA problem (94) and the LLNA problem (95) were both solved for values of $\lambda \in \{0, 0.01, 0.05, 0.10, 0.20, \ldots, 0.90, 0.95, 0.99, 1.0\}$. Figures 21 - 23 display results averaged over 5 noise vectors $p \in R^m$ with elements sampled from a normal distribution with mean = 0, standard deviation = 1. The average $\|p\|_1 = 21.1008$ and $\|b\|_1 = 20.1777$.

In Figure 21 we plot averages of $\|Ax(\lambda) - b - p\|_1$ for the various values of λ, measuring how "well" the PLNA and LLNA solutions solve the corrupted observed linear system. Also plotted is the average of $\|Ax(ls) - b - p\|_1$, measuring how "well" the least squares solution (Algorithm 4.5.1) solves the observed system $Ax = b + p$. As can be proved, the PLNA and LLNA errors are a non–decreasing functions of λ and are worse than the corresponding least squares error. However on the true system the results are reversed. See next paragraph.

In Figure 22 we plot averages of $\|Ax(\lambda) - b\|_1$ for both PLNA and LLNA for various values of λ, measuring how "well" the PLNA and LLNA solutions solve the true linear system. Also plotted is the average of $\|Ax(ls) - b\|_1$, measuring how "well" the least squares solution (Algorithm 4.5.1) solves $Ax = b$.

In Figure 23 we compare averages of 1-norm distances from the true solution x^* (90) to the PLNA and LLNA solutions $x(\lambda)$ and the averages of 1-norm distances from x^* to the least squares solution $x(ls)$. Recall that the true solution x^* is such that $Ax^* = b$. Note that for $\lambda \geq 0.01$, the PLNA and LLNA distances are smaller than the least squares distance. For $\lambda \approx 1$, $x(\lambda) \approx 0$ and even though $\|x(\lambda) - x^*\|_1$ is small, this solution is poor from a signal recovery point of view since the zero vector gives the worst discrepancy between the true signal and the recovered signal at 26 discrete points (see Figure 22).

In Figure 24 we plot the true signal, the observed signal and the signal recovered by solving, for one noise vector p, PLNA (94) with $\lambda = 0.30$ and LLNA (95) for $\lambda = 0.80$. Figure 25 displays the true signal, the observed signal and signal recovered for the same problem by least squares (96) solved by Algorithm 4.5.1. This is probably the most significant result. The signal recovered by both PLNA and LLNA is considerably

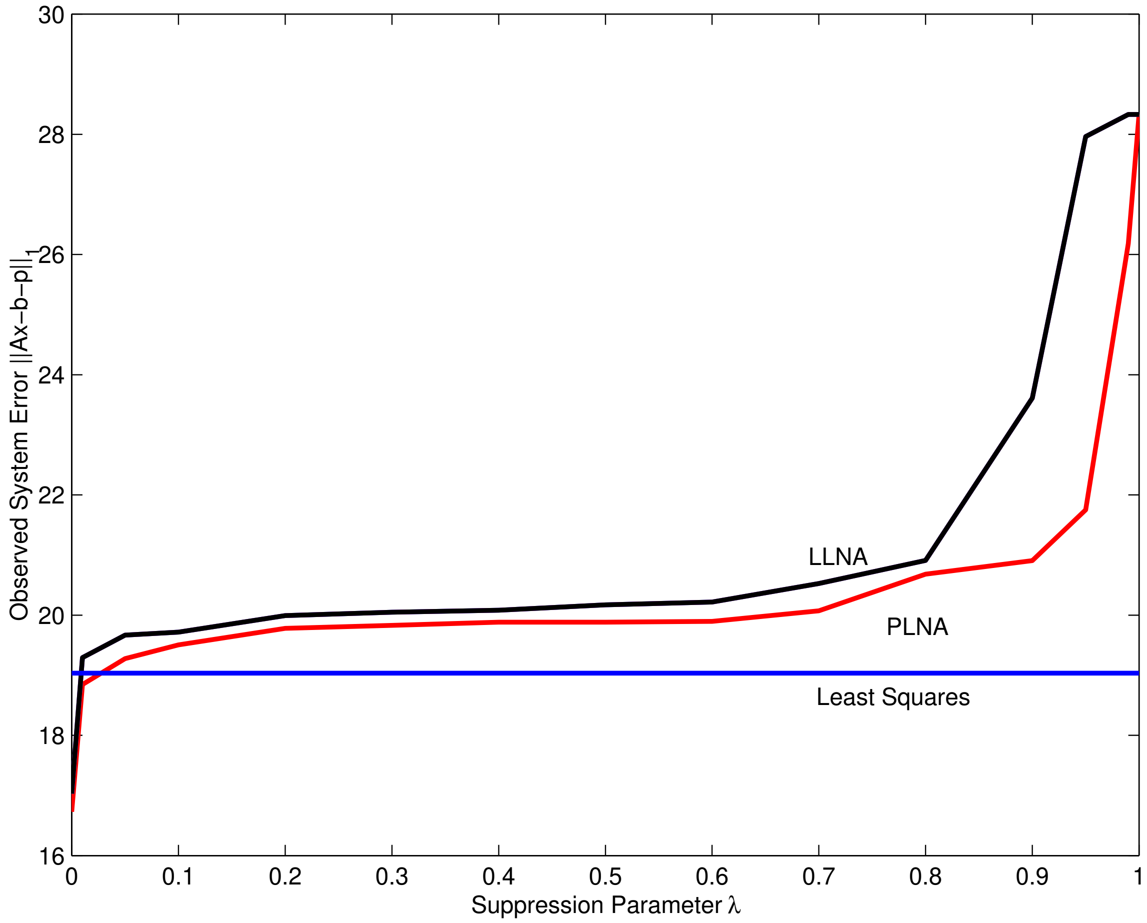


Figure 21: Average $\|Ax(\lambda) - b - p\|_1$ versus λ, where $x(\lambda)$ is a PLNA solution (94) in the curve marked PLNA and is an LLNA solution of (95) for the curve marked LLNA, compared with average $\|Ax(ls) - b - p\|_1$, where $x(ls)$ is the least squares solution (96) by Algorithm 4.5.1. The results are averaged over 5 noise vectors p. The PLNA and LLNA solutions were computed for values of $\lambda = 0, 0.01, 0.05, 0.10, 0.20, \ldots, 0.90, 0.95, 0.99, 1$.

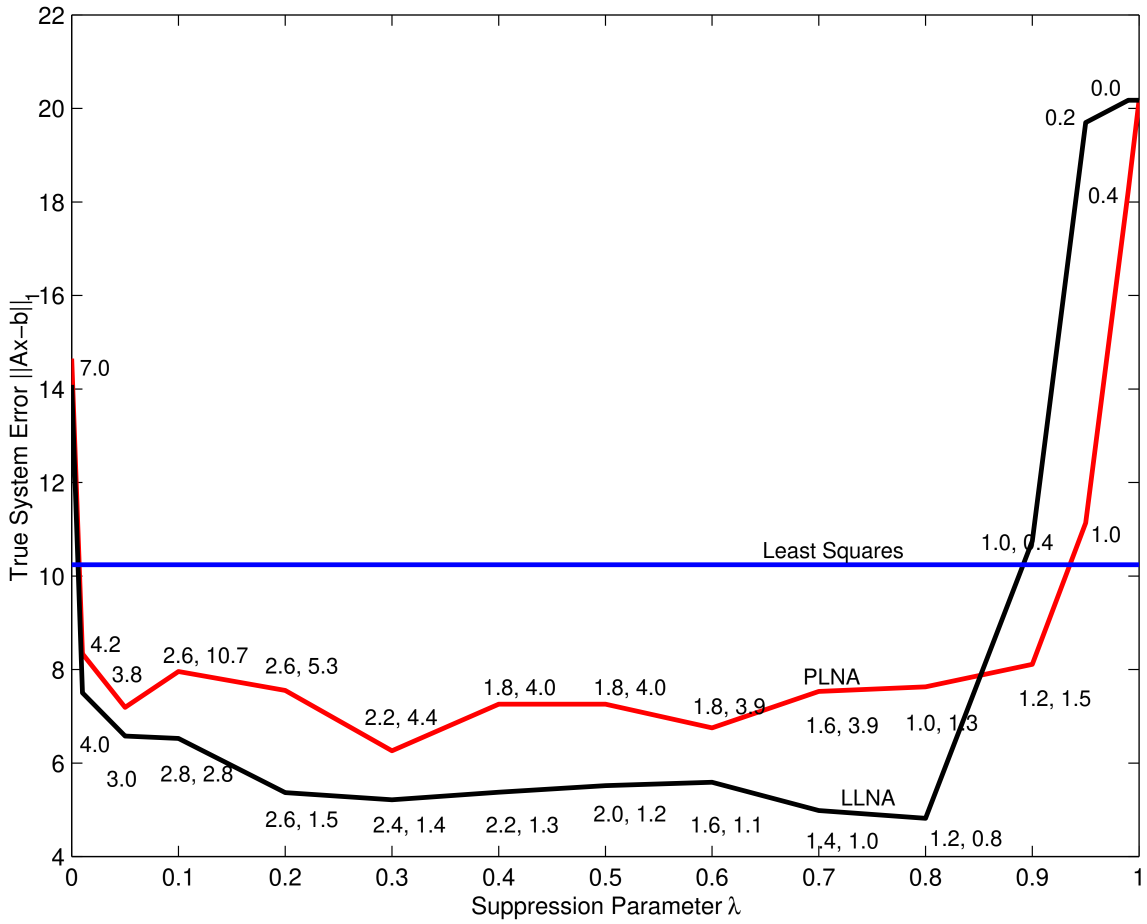


Figure 22: Average $\|Ax(\lambda) - b\|_1$ versus λ, where $x(\lambda)$ is a PLNA solution (94) in the curve marked PLNA and is an LLNA solution of (95) for the curve marked LLNA, compared with the average $\|Ax(ls) - b\|_1$, where $x(ls)$ is the least squares solution (96) solved by Algorithm 4.5.1. These results are averaged over 5 noise vectors p. The PLNA and LLNA solutions were computed for values of $\lambda = 0, 0.01, 0.05, 0.10, 0.20, \ldots, 0.90, 0.95, 0.99, 1$. Numbers above/below the curves labelled "PLNA" and "LLNA" at various values of λ indicate the average number of nonzero elements in $x(\lambda)$ and when followed by a second number, that number denotes $\|x(\lambda)\|_1$.

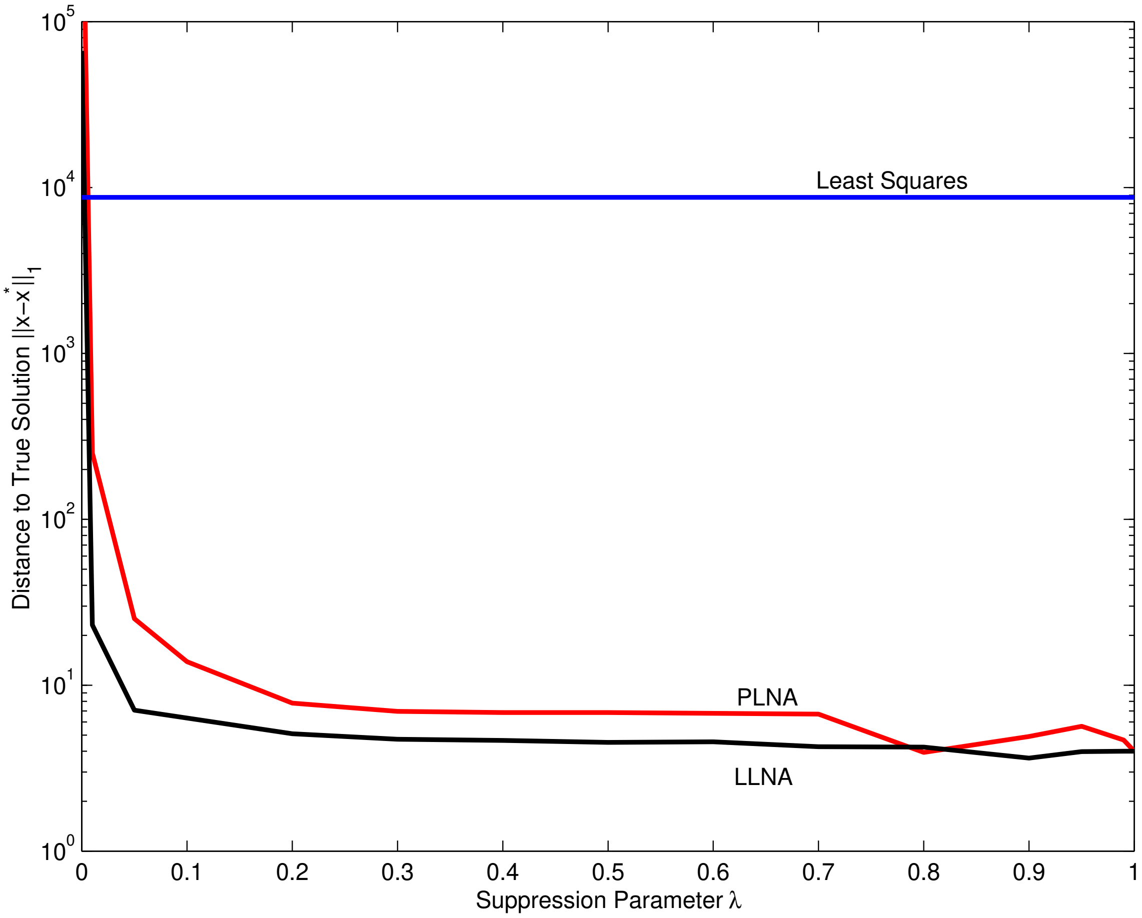


Figure 23: Average $\|x(\lambda) - x^*\|_1$ versus λ, where $x(\lambda)$ is a PLNA solution (94) in the curve marked PLNA and is an LLNA solution of (95) for the curve marked LLNA, compared with the average $\|x(ls) - x^*\|_1$, where $x(ls)$ is the least squares solution (96) solved by Algorithm 4.5.1. The true solution x^* (90) is such that $Ax^* = b$. The PLNA and LLNA solutions were computed for values of $\lambda = 0, 0.01, 0.05, 0.10, 0.20, \ldots, 0.90, 0.95, 0.99, 1$.

closer to the the true signal than that obtained by the least squares solution.

4.5.2 Comparison of PLNA, LLNA and Combinatorial Search

In this section, we reformulate our PLNA problem so that the solution $x(\lambda)$ has a fixed number of nonzero elements, for $k \in \{1, 2, \ldots, n\}$:

$$(x(\lambda), y(\lambda), z(\lambda)) \in \arg\min_{(x,y,z)\in R^{n+m+n}} \left\{ (1-\lambda)e'y + \lambda e'(e - \varepsilon^{-\alpha z}) \;\middle|\; \begin{array}{c} -y \leq Ax - b - p \leq y, \\ -z \leq x \leq z, \\ \#\text{ of nonzero elements of } x = k \end{array} \right\},$$
$$\lambda \in [0, 1). \quad (99)$$

We also formulate the LLNA similarly as follows:

$$(x(\lambda), y(\lambda), z(\lambda)) \in \arg\min_{(x,y,z)\in R^{n+m+n}} \left\{ (1-\lambda)e'y + \lambda e'z \;\middle|\; \begin{array}{c} -y \leq Ax - b - p \leq y, \\ -z \leq x \leq z, \\ \#\text{ of nonzero elements of } x = k \end{array} \right\},$$
$$\lambda \in [0, 1). \quad (100)$$

Similarly, for $k \in \{1, 2, \ldots, n\}$, the combinatorial search solution x_c is obtained by solving:

$$x_c \in \arg\min_{x\in R^n} \left\{ \|Ax - b - p\|_1 \;\middle|\; \#\text{ of nonzero elements of } x = k \right\}. \quad (101)$$

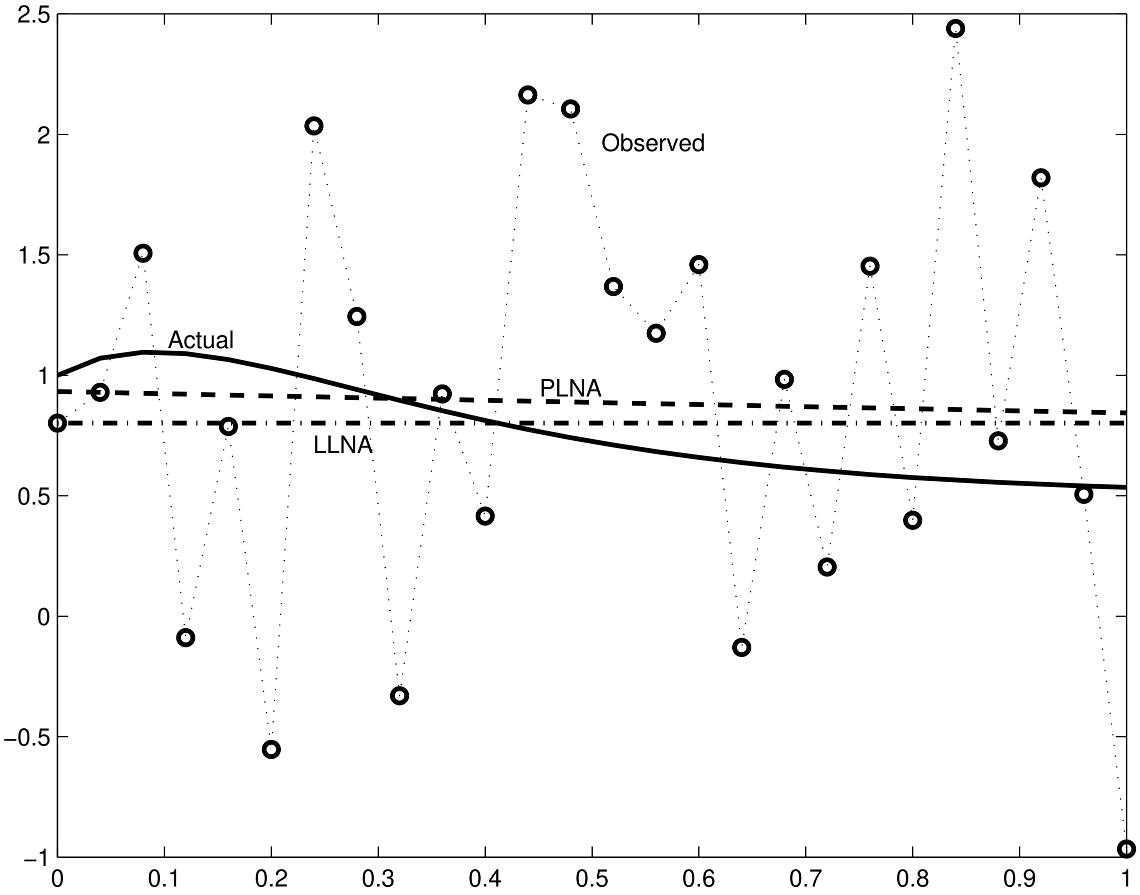


Figure 24: Dashed curves are the recovered signal $\hat{g}(t)$ with coefficient vector $x(\lambda)$ determined by (94) with $\lambda = 0.3$ and $\|Ax(\lambda) - b\|_1 = 4.7410$ for PLNA, and by (95) with $\lambda = 0.8$ and $\|Ax(\lambda) - b\|_1 = 4.7076$ for LLNA. Solid curve is the true signal $g(t)$. Circles are the observed signal $\tilde{g}(t_i)$ sampled at discrete times and the dashed curves are the recovered signals.

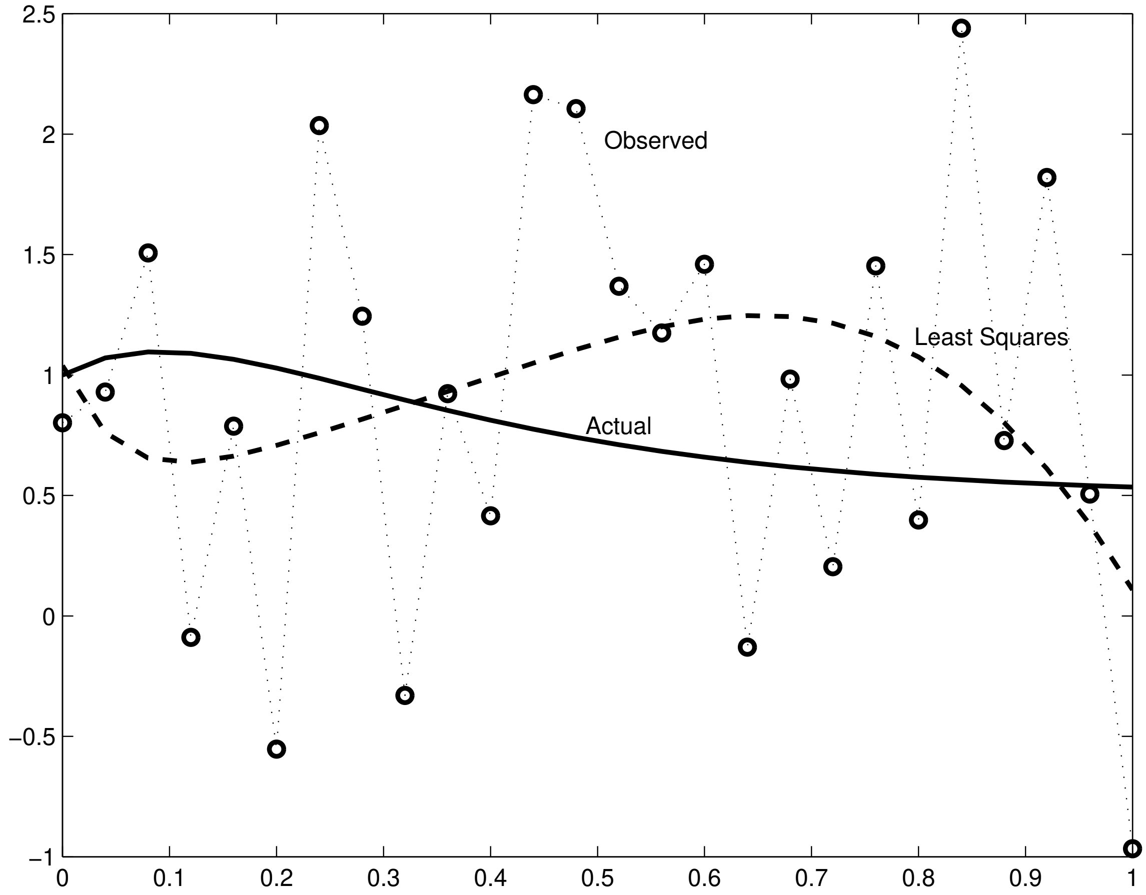


Figure 25: Dashed curve is the recovered signal $\hat{g}(t)$ with coefficient vector $x(ls)$ determined by least squares solution (96) solved by Algorithm 4.5.1. Note: $\|Ax(ls) - b\|_1 = 8.9733$. Solid curve is the true signal $g(t)$. Circles are the observed signal $\tilde{g}(t_i)$ sampled at discrete times and the dashed curves are the recovered signals.

Notice that x_c is determined by enumerating all subsets of size k of a set of n elements, or $\binom{n}{k}$ subsets. This is a rather expensive procedure computationally requiring two orders of magnitude more time than PLNA and LLNA.

Remark 4.5.2 *The following observation regarding problems (99) and (100) is due to R. R. Meyer [114]. We assume that problems (99) and (100) are well-posed (i.e. have optimal solutions). Alternatively, we could modify problems (99) and (100) requiring the number of nonzeros in the solution x to be less than or equal to k, instead of requiring the number of nonzeros to be equal to k. For instance, consider the following example:*

$$A = 1,\ b = p = 0,\ k = 1.$$

The linear system is simply $x = 0$ and a solution with $k = 1$ nonzeros with smallest residual does not exist.

Figure 26 displays results averaged over 5 noise vectors $p \in R^m$ with elements sampled from a normal distribution with mean = 0, standard deviation = 1 (average $\|p\|_1 = 21.1008$, $\|b\|_1 = 20.1777$). Plotted are averages of $\|Ax(\lambda) - b - p\|_1$ and $\|Ax_c - b - p\|_1$ for each k measuring how "well" the PLNA, LLNA and combinatorial solutions solve the observed system. Also plotted are averages of $\|Ax(\lambda) - b\|_1$ and $\|Ax_c - b\|_1$ for each k, measuring how "well" the solutions solve the true system.

Figure 27 displays the average 1-norm distance between x^* of (90) and the solutions obtained by PLNA, LLNA and combinatorial search. The averages are over 5 noise vectors p.

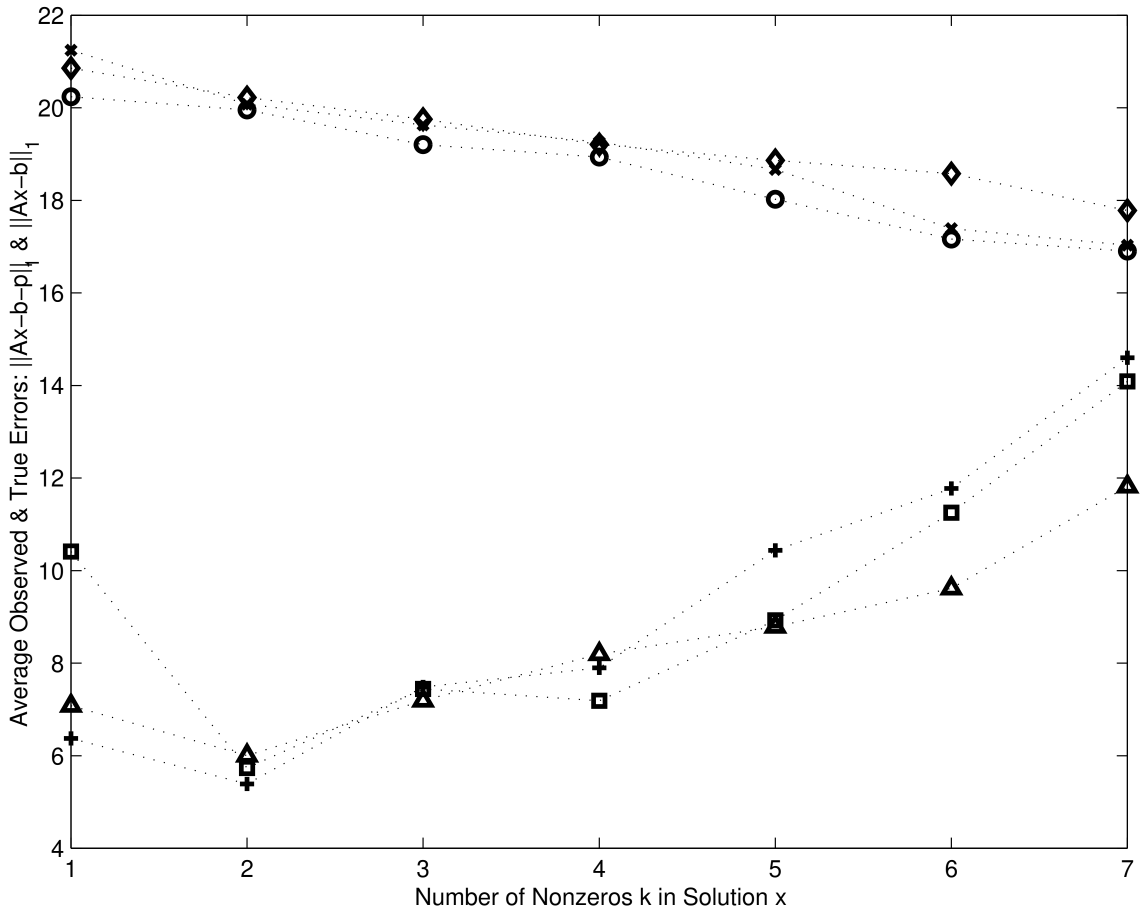


Figure 26: Comparison of PLNA (99) and LLNA (100) with combinatorial search (101). Average $\|Ax(\lambda) - b - p\|_1$ is '$\mathbf{x}$' for PLNA and '$\diamond$' for LLNA. Average $\|Ax_c - b - p\|_1$ is '$\mathbf{o}$'. Average $\|Ax(\lambda) - b\|_1$ is '$\Box$' for PLNA and $\triangle$ for LLNA. Average $\|Ax_c - b\|_1$ is '$+$' for combinatorial solution x_c.

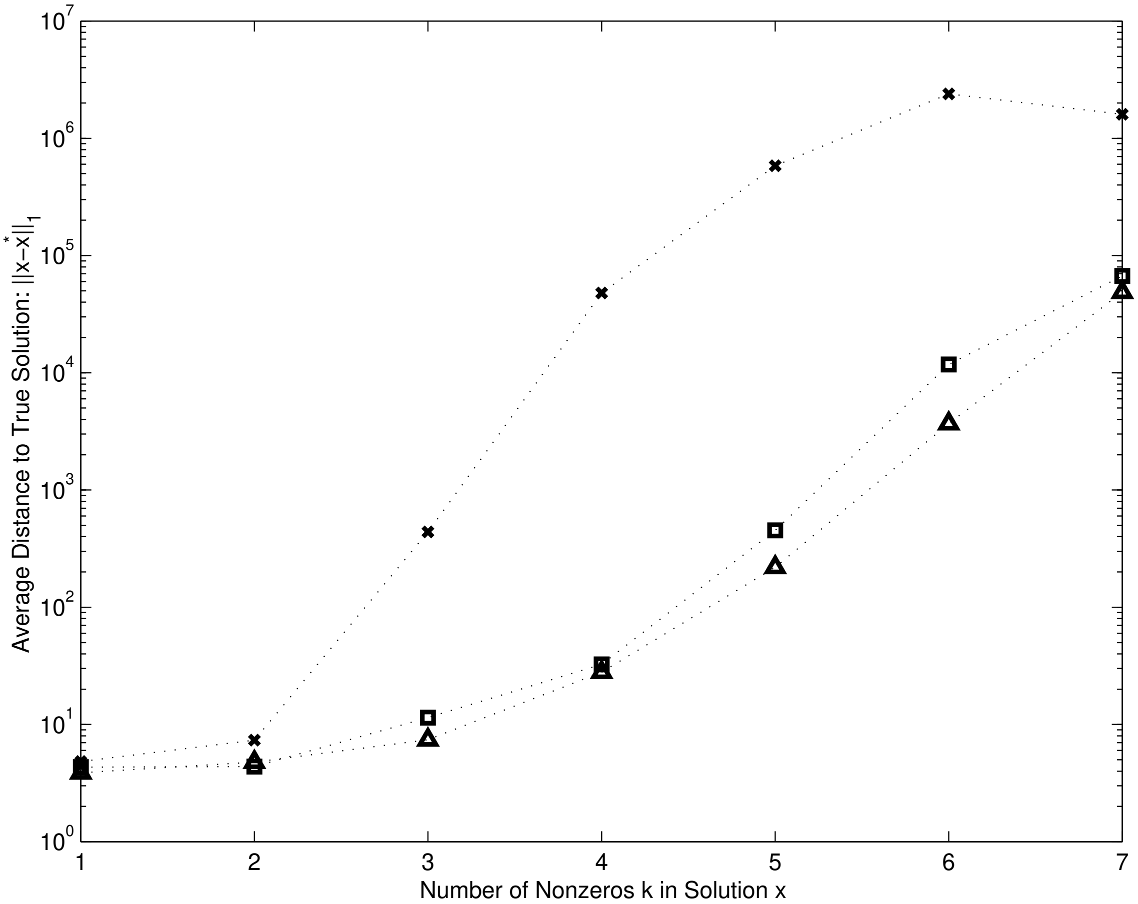


Figure 27: Comparison of PLNA (99) and LLNA (100) with combinatorial search (101). Average $\|x_c - x^*\|_1$ is '$\mathbf{x}$'. Average $\|x(\lambda) - x^*\|_1$ is '$\square$' for PLNA and $\triangle$ for LLNA. The true solution x^* is such that $Ax^* = b$.

Figure 28, which for convenience duplicates Figure 24, displays the true signal, the observed signal and the signal recovered by solving PLNA (94) for the value of $\lambda = 0.30$ and the signal recovered by LLNA (95) for $\lambda = 0.8$. Figure 29 displays the true signal, the observed signal and signal recovered by combinatorial search solution x_c of (101) for $k = 2$.

4.5.3 Observations

We make the following observations with respect to the comparison between the PLNA, LLNA solutions and least squares solutions.

1. For all values of $\lambda \geq 0.05$ tested, the average observed system (training set) residual $\|Ax(ls) - b - p\|_1$ was strictly less than the average $\|Ax(\lambda) - b - p\|_1$ for both PLNA and LLNA. The least squares Algorithm 4.5.1 for solving (96) produced "better" solutions to the observed system $Ax = b + p$. See Figure 21. However......

2. For values of $\lambda \in [0.01, 0.90]$ tested, the PLNA average true system (testing set) residual $\|Ax(\lambda) - b\|_1$ was strictly less than the average $\|Ax(ls) - b\|_1$ indicating that PLNA produced "better" solutions to the true system $Ax = b$ in comparison with least squares. For values of $\lambda \in [0.01, 0.80]$ tested, the average true system residual with solutions determined by LLNA was also strictly less than the corresponding least squares true system residuals. See Figure 22. PLNA with $\lambda = 0.3$ and an average of 2.2 nonzero terms achieved an error reduction of 38.85% over the corresponding error obtained by the least squares solution.

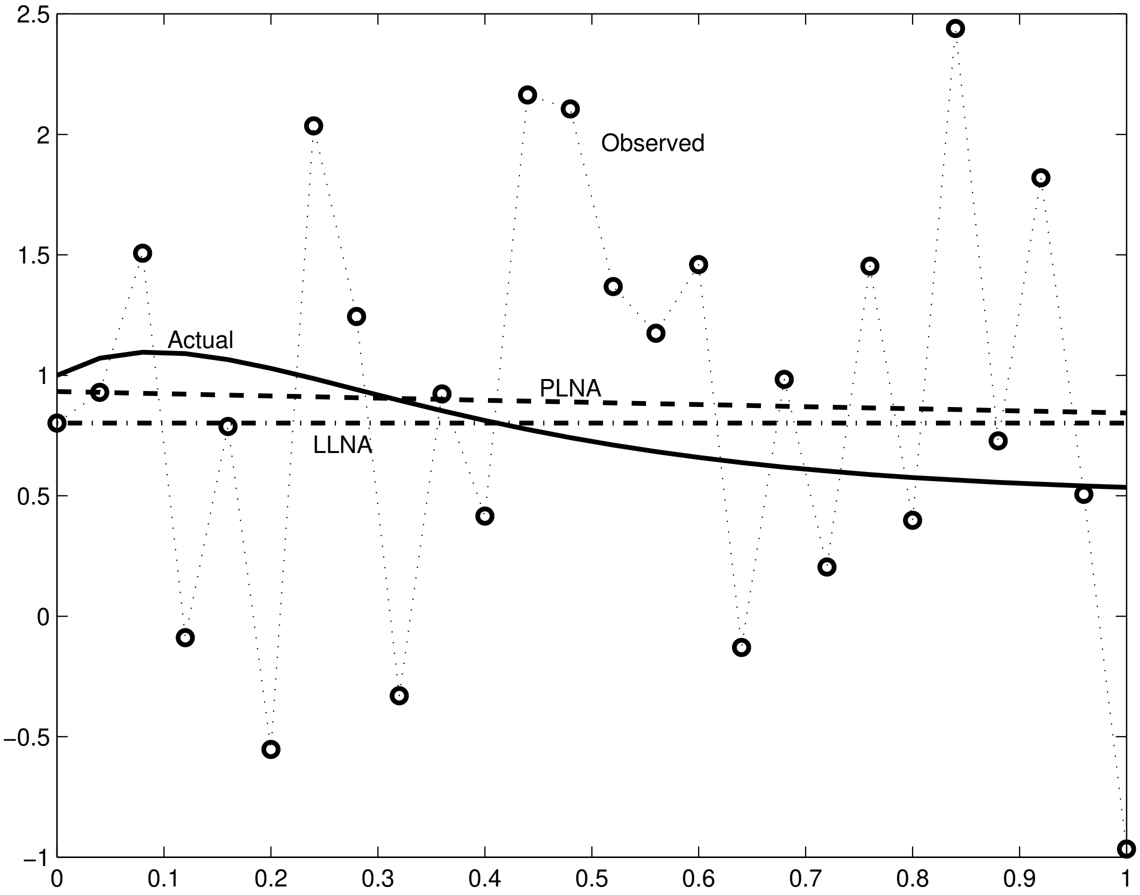


Figure 28: Dashed curves are the recovered signal $\hat{g}(t)$ with coefficient vector $x(\lambda)$ determined by (94) with $\lambda = 0.3$ and $\|Ax(\lambda) - b\|_1 = 4.7410$ for PLNA, and by (95) with $\lambda = 0.8$ and $\|Ax(\lambda) - b\|_1 = 4.7076$ for LLNA. Solid curve is the true signal $g(t)$. Circles are the observed signal $\tilde{g}(t_i)$ sampled at discrete times and the dashed curves are the recovered signals.

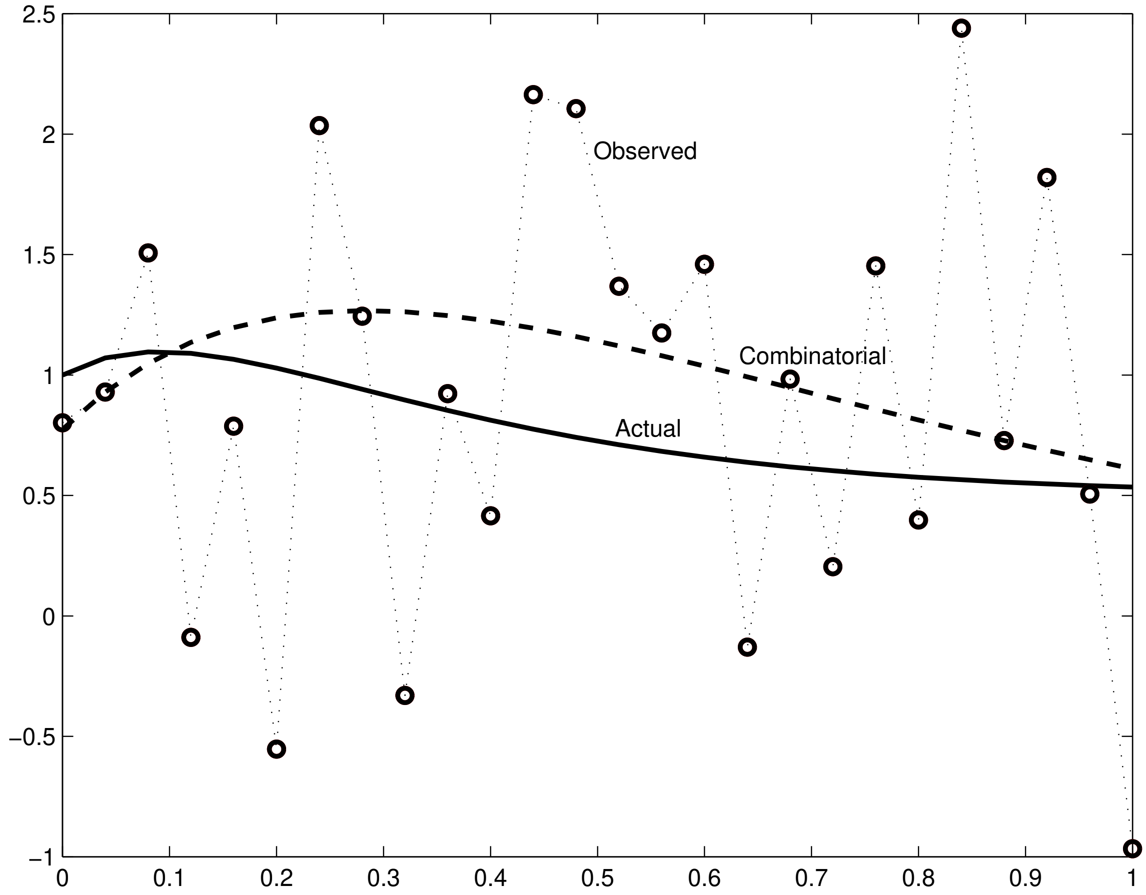


Figure 29: Dashed curve is the recovered signal $\hat{g}(t)$ with coefficient vector x_c determined by combinatorial search with $k = 2$ (101). Note: $\|Ax_c - b\|_1 = 6.7826$. Solid curve is the true signal $g(t)$. Circles are the observed signal $\tilde{g}(t_i)$ sampled at discrete times and the dashed curves are the recovered signals.

LLNA with $\lambda = 0.8$ produced an average 1-norm true system residual that was 52.98% less than the least squares residual.

3. For values of $\lambda > 0.1$ tested, the average $\|x(\lambda) - x^*\|_1$, determined by both PLNA and LLNA, was 2 orders of magnitude less than the average $\|x(ls) - x^*\|$. Hence the PLNA and LLNA solutions were "closer" to recovering the true signal $g(t)$ (87). See Figure 23.

4. Figures 24 and 25, show the most significant comparison between PLNA, LLNA and least squares: A much more accurate recovery of the true signal by both PLNA and LLNA than by least squares.

We note the following with respect to the comparison between the PLNA, LLNA solutions and the solutions obtained by combinatorial search.

1. For $k = 3, 4, 5, 6, 7$, the average PLNA $\|Ax(\lambda) - b\|_1$ was strictly less than the average $\|Ax_c - b\|_1$. For $k = 1, 2$, the average PLNA $\|Ax(\lambda) - b\|_1$ was less than or equal to 1.634 times the average $\|Ax_c - b\|_1$. For $k = 3, 5, 6, 7$, the average LLNA $\|Ax(\lambda) - b\|_1$ was strictly less than the corresponding average true system residual with the combinatorial solutions. For $k = 1, 2, 4$, the average LLNA $\|Ax(\lambda) - b\|_1$ was less than or equal to 1.114 times the corresponding average $\|Ax_c - b\|$. See Figure 26.

2. For $k \geq 3$, the average $\|x(\lambda) - x^*\|_1$, for both PLNA and LLNA, was strictly less than the average $\|x_c - x^*\|_1$ by orders of magnitude. For $k = 0, 1, 2$, average $\|x(\lambda) - x^*\|_1$ was less than or equal to average $\|x_c - x^*\|_1$. See Figure 27.

3. The minimum over $k = 1, \ldots, 7$ of the true system 1-norm residual of 5.3867 occurs for $k = 2$ with the solution obtained by combinatorial search. The true system residual for PLNA with $k = 2$ is 5.7330 and the true system residual for LLNA is 6.0022. We note that when computing the PLNA and LLNA solutions for $k = 2$, the first value of λ found (by a bisection search) such that the solution has 2 nonzero elements was chosen. This fact accounts for the discrepancy between the true system residuals in Figure 26 and Figure 22.

4. Figures 28 and 29 show recovery of the true signal by both PLNA and LLNA which is as good or even better than the recovered signal by a lengthy combinatorial search.

The time needed by each approach to compute a solution was determined by performing a single run on a Sun SparcStation 20 with 96 megabytes of memory running MATLAB 5.1, using the commands "tic" and "toc" [110]. All linear programs were solved with CPLEX [48] interfaced with MATLAB. Solving the PLNA problem with $\lambda = 0.5$ with initial point (93) and $\alpha = 5$ took 0.4603 seconds. Solving the LLNA problem with $\lambda = 0.5$ took 0.1978 seconds. Determining the least squares solution by Algorithm 4.5.1 with $\tau = 0.0001$ took 0.0224 seconds. Determining the solution by combinatorial search with $k = 3$ took 13.2008 seconds.

Solutions computed by PLNA and LLNA were at most superior or at least comparable to those obtained by combinatorial search (101), yet needing two orders of magnitude less time to compute.

In this chapter, we have proposed a theoretically justifiable fast finite algorithm has been proposed for solving linear systems corrupted by noise or errors in measurement.

The parsimonious approach (PLNA) attempts to set to zero as many components of the solution vector as possible while minimizing the residual error of the corrupted system, whereas the least norm approach (LLNA) minimizes the norm of the solution as well as the residual. Numerical evidence indicates that both these two approaches lead to solutions with many zero components, and that such solutions may be closer by orders of magnitude to the solution of the underlying uncorrupted system than other solutions of the corrupted system obtained by either least squares or even by a time-consuming combinatorial search for a solution with a minimal number of nonzero components. It is interesting to note that parametricly minimizing the norm of the solution leads also to suppression of its components, and conversely parametrically suppressing components of the solution also leads to a solution with a reduced norm. Most importantly, PLNA and LLNA recover a much more accurate signal than that obtained by least squares and much faster than that obtained by a lengthy combinatorial search.

Chapter 5

Conclusions

This thesis has investigated the application of mathematical programming methods to problems arising in machine learning. Methods for solving these machine learning problems are effective data mining tools for use in the KDD process.

5.1 Machine Learning: Supervised Learning

We have described a novel approach to the problem of feature selection by introducing a parametric objective function in a mathematical program that attempts to separate data by utilizing as few features as possible. This can also be achieved by finding a minimum support solution [106] to the robust linear program (7).

We have also presented a new and simple realization of the complexity-reducing Optimal Brain Damage procedure that consists of solving two linear programs (Algorithm 2.1.6). Mathematical programming approaches addressing the classification task are the Feature Selection Sigmoid (FSS) (12), Feature Selection Concave (FSV) (13) and Feature Selection Bilinear (FSB) (16) problems. These techniques in addition to the Optimal Brain Damage (OBD) Algorithm 2.1.6, produce classifiers with improved generalization ability when applied to the Wisconsin Prognostic Breast Cancer classification task, while reducing the number of problem features used. All but the

OBD Algorithm 2.1.6 either improve generalization or produce classifiers with "equivalent" performance on the Ionosphere classification task, while utilizing a fraction of the original problem features. The FSS problem (12) is solved by iterative quadratic programming [65, 74, 67]. The FSV problem (13) is solved by solving a sequence of linear programs via the Successive Linearization Algorithm 2.1.3, which is a general tool for solving problems with a concave objective over a polyhedral region [103]. The FSV problem (16) is solved by [9, Algorithm 2.1].

Classifiers produced by the support vector machine [161, 33] (SVM) approach were presented. Minimizing an upper bound on generalization performance [161, 33] motivates the SVM formulation which attempts to separate the training data $\mathcal{A}$ and $\mathcal{B}$ with a maximum margin. When the margin of separation is measured in the ∞-norm, the 1-norm appears as a penalty on the weight vector $w \in R^n$ in the SVM objective yielding a linear program (35). If the margin is measured in the 1-norm, the ∞-norm appears as a penalty on the weight vector again yielding a linear program (34). When the margin is measured in the 2-norm, one usually appends the 2-norm squared of w in the SVM objective giving rise to the minimization of a convex quadratic function over a polyhedral region (36). The SVM classifiers were experimentally evaluated along with classifiers obtained by solving the FSV problem (13) and the RLP problem (7), which does not explicitly force feature suppression. Classifiers obtained by solving FSV (13) and SVM 1-norm (35) exhibit feature suppression and have comparable generalization performance on six pubicly available real world data sets tested. The classifiers obtained by solving the FSV problem (13) suppressed more problem features than the corresponding SVM 1-norm classifiers (35).

Another class of problems investigated were minimum support solutions of linear equalities. This was motivated by machine learning regression problems that can be formulated as computing a solution to a linear system of equation $Ax = b$, where the vector b is corrupted by noise and a minimum support solution x is likely to lead to improved generalization. The PLNA approach (76) attempts to set to zero as many components of the solution vector as possible while minimizing the residual error of the linear system. The least norm approach, LLNA (77), minimizes the norm of the solution as well as the residual. Numerical evidence indicates that both of these approaches lead to solutions with many zero components, and that such solutions may be closer by orders of magnitude to the solution of the underlying uncorrupted system than other solutions of the corrupted system obtained by either least squares or even by a time-consuming combinatorial search. It is interesting to note that parametrically minimizing the 1-norm of the solution leads also to suppression of its components, and conversely parametrically suppressing components of the solution also leads to a solution with reduced norm. Most importantly, PLNA and LLNA recover a much more accurate signal than that obtained by least squares and much faster than that obtained by a lengthy combinatorial search on the example in Section 4.5.2. The continuous version of the PLNA problem consists of minimizing a concave function over a polyhedral set and is efficiently solved via the Successive Linearization Algorithm 4.4.1, similar to the one used to solve the FSV problem (13).

5.2 Machine Learning: Unsupervised Learning

We have described an approach for assigning m data points in R^n into k clusters based on a simple concave minimization model. Although a global solution to the problem cannot be guaranteed, the finite and simple k-Median Algorithm 3.1.3 is able to compute useful clustering solutions when initialized using a random reset strategy [55]. The utility of the k-Median Algorithm 3.1.3 and k-Mean Algorithm 3.1.4 as data mining tools were evaluated over moderately sized medical and other datasets. Both algorithms were able to identify three populations in both the Wisconsin Prognostic Breast Cancer Database (WPBC) and the SEER database with distinct survival characteristics. The k-Median Algorithm 3.1.3 is preferable to the k-Mean Algorithm 3.1.4 when attempting to cluster a dataset containing numerous outliers as the computation of the median is more robust than that of the mean. We have also shown finite termination of the k-Median Algorithm 3.1.3 to a locally optimal solution.

A new clustering algorithm based on minimizing the sum of squared distances of points to a closest cluster plane instead of the conventional closest cluster center was introduced. The k-Plane Algorithm 3.2.1 computes cluster planes by solving an eigenvalue problem for each of the k clusters at each iteration. The utility of this scheme as a data mining tool was exhibited on the WPBC and SEER datasets as clusters with distinct survival characteristics were identified. Other advantages to having the k-Plane algorithm in the data miner's toolbox include the ability to cluster points that fall naturally into a subspace of the original data space and hence may be better approximated by a plane.

5.3 Massive Dataset Issues

The LPC Algorithm 2.3.1 is significant in its own right as a linear programming decomposition algorithm, enabling the computation of solutions to linear programs with massive constraints. The LPC Algorithm was tested on an expanded version of the WOODW dual problem from the NETLIB repository [1]. It was motivated and further tested on the classification task over massive datasets that may not fit into resident machine memory. The algorithm uses support vector ideas by keeping only essential data points needed for determining a separating plane. The algorithm deals with small chunks of data at a time and is guaranteed to terminate in a finite number of steps at an optimal solution when all subproblems are solvable. The algorithm can be easily parallelized by splitting the data among many processors and sharing only support vectors among them. Extremely large datasets could effectively be processed on a network of PCs or workstations.

The LPC Algorithm 2.3.1 can be used to solve the linear programs of the Successive Linearization Algorithm 2.1.3 and Algorithm 4.4.1. Thus the FSV (13) and PLNA (76) can be effectively addressed over massive datasets.

Issues still remaining include initialization and local versus global optimality of the concave minimization problems and the bilinear problems, applied utility of using different norms to measure the margin of separation in the support vector machine formulation and a theoretical link between minimizing the number of nonzero elements of a solution versus minimizing the norm of the solution.

Useful solutions were computed via the Successive Linearization Algorithms 2.1.3 and 4.4.1 and the k-Median Algorithm 3.1.3 with random initializations. Knowledge about the particular problem domain in which these algorithms are applied may lead to a better initialization procedure. We note that issues regarding clustering initialization are provided in [18, 59].

Analysis of the classifiers computed by the support vector machine approach utilizing different norms to measure the margin of separation is needed. If knowledge of the problem domain is available, the results of this analysis could aid a user in *a priori* choosing an "appropriate" norm for their particular classification problem. On very massive classification tasks, the user may not have the luxury of "evaluating" classifiers computed using different norms.

In addition a theoretical link between the minimization of the number of nonzero elements of a vector and the norm of the vector is needed (see Sections 2.1.2,2.2.1 and 4.2). Initial work has been done [114].

In this work we have formulated machine learning tasks as mathematical programs and investigated algorithms which compute solutions to these problems. We have exhibited the utility of these approaches over real-wold datasets and propose the addition of these methods as data mining tools to aid in extracting useful "knowledge" from possibly datasets. We have further enabled the linear-programming-based methods to efficiently be applied to massive datasets often occurring in many KDD applications.

Bibliography

[1] The NETLIB test problem set. www.enseeiht.fr/netlib/index.html.

[2] T. J. Abatzoglou, J. M. Mendel, and G. A. Harada. The constrained total least squares technique and its application to harmonic superposition. *IEEE Transactions on Signal Processing*, 39:1070–1087, 1991.

[3] K. Al-Sultan. A Tabu search approach to the clustering problem. *Pattern Recognition*, 28(9):1443–1451, 1995.

[4] H. Almuallin and T. G. Deitterich. Learning with many irrelevant features. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 547–552, 1991.

[5] K. P. Bennett. Decision tree construction via linear programming. In M. Evans, editor, *Proceedings of the 4th Midwest Artificial Intelligence and Cognitive Science Society Conference*, pages 97–101, Utica, Illinois, 1992.

[6] K. P. Bennett and J. A. Blue. A support vector machine approach to decision trees. Department of Mathematical Sciences Math Report No. 97-100, Rensselaer Polytechnic Institute, Troy, NY 12180, 1997. http://www.math.rpi.edu/˜bennek/.

[7] K. P. Bennett and O. L. Mangasarian. Neural network training via linear programming. In P. M. Pardalos, editor, *Advances in Optimization and Parallel Computing*, pages 56–67, Amsterdam, 1992. North Holland.

[8] K. P. Bennett and O. L. Mangasarian. Robust linear programming discrimination of two linearly inseparable sets. *Optimization Methods and Software*, 1:23–34, 1992.

[9] K. P. Bennett and O. L. Mangasarian. Bilinear separation of two sets in n-space. *Computational Optimization & Applications*, 2:207–227, 1993.

[10] M. W. Berry, S. T. Dumais, and G. W. O'Brein. Using linear algebra for intelligent information retrieval. *SIAM Review*, 37:573–595, 1995. http://www.cs.utk.edu/~berry.

[11] C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, New York, 1995.

[12] V. Blanz, B. Schölkopf, H. Bülthoff, C. Burges, V. Vapnik, and T. Vetter. Comparison of view-based object recognition algorithms using realistic 3d models. In *Artificial Neural Networks – ICANN96*, pages 251–256, Berlin, 1996. Springer Lecture Notes in Computer Science, Vol. 1112.

[13] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Occam's razor. *Information Processing Letters*, 24:377–380, 1987.

[14] M. Boddy and T. Dean. Decision-theoretic deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence*, 67(2):245–286, 1994.

[15] S. Bös. A realizable learning task which shows overfitting. In *Advances in Neural*

Information Processing Systems 8, pages 218–224, Cambridge, MA, 1996. The MIT Press.

[16] S. Bös and M. Opper. An exact description of early stopping and weight decay. Lab for Information Representation, RIKEN, Wako-shi Saitama 351-01, Japan. November 21, 1996. See http://www.bip.riken.go.jp/irl/boes/boes.html.

[17] L. Bottou and Y. Bengio. Convergence properties of the k-means algorithms. In *Advances in Neural Information Processing Systems -7-*, pages 585–592, Cambridge, MA, 1995. MIT Press.

[18] P. S. Bradley and U. M. Fayyad. Refining initial points for k-means clustering. In J. Shavlik, editor, *Proceedings of the Fifteenth International Conference on Machine Learning (ICML '98)*, pages 91–99, San Francisco, CA, 1998. Morgan Kaufmann. http://www.cs.wisc.edu/~paulb/papers.html.

[19] P. S. Bradley, U. M. Fayyad, and C. Reina. Scaling clustering to large databases. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining, KDD98*, 1998. http://www.cs.wisc.edu/~paulb/papers.html.

[20] P. S. Bradley, U. M. Fayyad, and C. Reina. Scaling EM clustering to large databases. Technical report, Microsoft Research, Redmond, WA, 1998. http://www.cs.wisc.edu/~paulb/papers.html.

[21] P. S. Bradley, Usama M. Fayyad, and O. L. Mangasarian. Data mining: Overview and optimization opportunities. Technical Report 98-01, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, January

1998. *INFORMS Journal on Computing*, submitted. ftp://ftp.cs.wisc.edu/math-prog/tech-reports/998-01.ps.Z.

[22] P. S. Bradley and O. L. Mangasarian. Feature selection via concave minimization and support vector machines. In J. Shavlik, editor, *Machine Learning Proceedings of the Fifteenth International Conference(ICML '98)*, pages 82–90, San Francisco, California, 1998. Morgan Kaufmann. ftp://ftp.cs.wisc.edu/math-prog/tech-reports/98-03.ps.Z.

[23] P. S. Bradley and O. L. Mangasarian. k-Plane clustering. Technical Report 98-08, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, August 1998. ftp://ftp.cs.wisc.edu/math-prog/tech-reports/98-08.ps.Z.

[24] P. S. Bradley and O. L. Mangasarian. Massive data discrimination via linear support vector machines. Technical Report 98-05, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, May 1998. ftp://ftp.cs.wisc.edu/math-prog/tech-reports/98-03.ps.Z.

[25] P. S. Bradley, O. L. Mangasarian, and J. B. Rosen. Parsimonious least norm approximation. *Computational Optimization and Applications*, (1):5–21, October 1998. ftp://ftp.cs.wisc.edu/math-prog/tech-reports/97-03.ps.Z.

[26] P. S. Bradley, O. L. Mangasarian, and W. N. Street. Clustering via concave minimization. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems -9-*, pages 368–374, Cambridge, MA, 1997. MIT Press. ftp://ftp.cs.wisc.edu/math-prog/tech-reports/96-03.ps.Z.

[27] P. S. Bradley, O. L. Mangasarian, and W. N. Street. Feature selection via mathematical programming. *INFORMS Journal on Computing*, 10(2):209–217, 1998. ftp://ftp.cs.wisc.edu/math-prog/tech-reports/95-21.ps.Z.

[28] E. J. Bredensteiner and K. P. Bennett. Feature minimization within decision trees. *Computational Optimizations and Applications*, 10:111–126, 1998.

[29] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth Publishing Company, Belmont, CA, 1983.

[30] C. E. Brodley and P. E. Utgoff. Multivariate decision trees. *Machine Learning*, 19(1):45–77, 1995.

[31] A. Brooke, D. Kendrick, and A. Meeraus. *GAMS: A User's Guide.* The Scientific Press, South San Francisco, CA, 1988.

[32] W. L. Buntine. Graphical models for discovering knowledge. In *Advances in Knowledge Discovery and Data Mining*, pages 59–81, Menlo Park, CA, 1996. AAAI Press.

[33] C. J. C. Burges. A tutorial on support vector machines. *Data Mining and Knowledge Discovery*, 2, 1998.

[34] C. J. C. Burges and B. Schölkopf. Improving the accuracy and speed of support vector machines. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems -9-*, pages 375–381, Cambridge, MA, 1997. MIT Press.

[35] C. L. Carter, C. Allen, and D. E. Henson. Relation of tumor size, lymph node status, and survival in 24,740 breast cancer cases. *Cancer*, 63:181–187, 1989. SEER: Surveillence, Epidemiology and End Results Program of the National Cancer Institute.

[36] R. Caruana and D. Freitag. Greedy attribute selection. In *Machine Learning: Proc 11th Intl Conf*, pages 28–36, New Brunswick, NJ, 1994. Morgan Kaufmann.

[37] T. Cavalier and B. Melloy. An iterative linear programming solution to the Euclidean regression model. *Computers and Operations Research*, 28:781–793, 1995.

[38] C. Y. Chang. Dynamic programming as applied to feature subset selection in a pattern recognition system. *IEEE Trans. Syst. Man Cybern.*, SMC-3:166–171, March 1973.

[39] P. Cheeseman and J. Stutz. Bayesian classification (autoclass): Theory and results. In *Advances in Knowledge Discovery and Data Mining*, pages 153–180, Menlo Park, CA, 1996. AAAI Press.

[40] Chunhui Chen and O. L. Mangasarian. Hybrid misclassification minimization. *Advances in Computational Mathematics*, 5(2):127–136, 1996. ftp://ftp.cs.wisc.edu/math-prog/tech-reports/95-05.ps.Z.

[41] S. S. Chen, D. L. Donoho, and M. A. Saunders. Atomic decomposition by basis pursuit. Technical Report 479, Department of Statistics, Stanford University, Stanford, California 94305, February 1996. Available by http://playfair.stanford.EDU/reports/chen_s/BasisPursuit.ps.Z.

[42] K. J. Cherkauer and J. W. Shavlik. Growing simpler decision trees to facilitate knowledge discoverey. In *Proceedings, Second International Conference on Knowledge Discovery and Data Mining*. AAAI Press, 1996.

[43] S.-J. Chung. NP-completeness of the linear complementarity problem. *Journal of Optimization Theory and Applications*, 60:393–399, 1989.

[44] V. Chvátal. *Linear Programming.* W. H. Freeman and Company, New York, 1983.

[45] F. Cordellier and J. Ch. Fiorot. On the Fermat-Weber problem with convex cost functionals. *Mathematical Programming*, 14:295–311, 1978.

[46] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273–279, 1995.

[47] R. W. Cottle, J.-S. Pang, and R. E. Stone. *The Linear Complementarity Problem.* Academic Press, New York, 1992.

[48] CPLEX Optimization Inc., Incline Village, Nevada. *Using the CPLEX(TM) Linear Optimizer and CPLEX(TM) Mixed Integer Optimizer (Version 2.0)*, 1992.

[49] G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8:101–111, 1960.

[50] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society B*, 39:1–38, 1977.

[51] P. A. Devijver and J. Kittler. *Pattern Recognition: A Statistical Approach.* Prentice–Hall, Inc, Englewood Cliffs, NJ, 1982.

[52] J. L. Devore. *Probability and Statistics for Engineering and the Sciences.* Wadsworth Publishing Company, Belmont, California, 1995. Fourth Edition.

[53] T. G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 1998. http://www.cs.orst.edu/˜tgd/cv/pubs.html.

[54] H. Drucker, C. J. C. Burges, L. Kaufman, A. Smola, and V. Vapnik. Support vector regression machines. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems -9-*, pages 155–161, Cambridge, MA, 1997. MIT Press.

[55] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* John Wiley & Sons, New York, 1973.

[56] U. Fayyad, D. Haussler, and P. Stolorz. KDD for science data analysis: Issues and examples. In *Proceedings, Second International Conference on Knowledge Discovery and Data Mining.* AAAI Press, 1996.

[57] U. Fayyad, G. Piatetsky-Shapiro, and Padhraic Smyth. Knowledge discovery and data mining: Towards a unifying framework. In *Proceedings, Second International Conference on Knowledge Discovery and Data Mining.* AAAI Press, 1996.

[58] U. M. Fayyad. Editorial. *Data Mining and Knowledge Discovery*, 1(1):5–10, 1997.

[59] U. M. Fayyad, C. Reina, and P. S. Bradley. Initialization of iterative refinement clustering algorithms. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining, KDD98*, 1998. http://www.cs.wisc.edu/~paulb/papers.html.

[60] D. Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2:139–172, 1987.

[61] E. W. Forgy. Cluster analysis of multivariate data: Efficiency versus interpretability of classifications. *Biometric Soc. Meetings*, Riverside CA (Abstract in *Biometrics* 21, No. 3, 768), 1965.

[62] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3:95–110, 1956.

[63] K. Fukunaga. *Statistical Pattern Recognition.* Academic Press, NY, 1990.

[64] P. P. Gallo. Consistency of regression estimates when some variables are subject to error. *Comm. Statist. Theory Methods*, 11:973–983, 1982.

[65] U. M. Garcia Palomares and O. L. Mangasarian. Superlinearly convergent Quasi–Newton algorithms for nonlinearly constrained optimization problems. *Mathematical Programming*, 11:1–13, 1976.

[66] P. E. Gill. Private communication, July 16 1998.

[67] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization.* Academic Press, London, 1981.

[68] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting stock problem. *Operations Research*, 9:849–859, 1961.

[69] A. A. Giordano. *Least Square Estimation With Applications to Digital Signal Processing*. John Wiley & Sons, New York, 1985.

[70] L. J. Gleser. Estimation in a multivariate "errors in variables" regression model: Large sample results. *Annals of Statistics*, 9:24–44, 1981.

[71] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison–Wesley, Reading MA, 1989.

[72] G. H. Golub and C. F. Van Loan. An analysis of the total least squares problem. *SIAM Journal on Numerical Analysis*, 17:883–893, 1980.

[73] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The John Hopkins University Press, Baltimore, Maryland, 1983.

[74] S.-P. Han. Superlinearly convergent variable metric algorithms for general nonlinear programming problems. *Mathematical Programming*, 11:263–282, 1976.

[75] M. H. Hassoun. *Fundamentals of Artificial Neural Networks*. MIT Press, Cambridge, MA, 1995.

[76] D. Heckerman. Bayesian networks for knowledge discovery. In *Advances in Knowledge Discovery and Data Mining*, pages 273–305, Menlo Park, CA, 1996. AAAI Press.

[77] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation*. Addison-Wesley, Redwood City, California, 1991.

[78] P. J. Huber. *Robust Statistics.* John Wiley, New York, 1981.

[79] S. Van Huffel, H. Park, and J. B. Rosen. Formulation and solution of structured total least norm problems for parameter estimation. *IEEE Transactions on Signal Processing*, 44:2464–2474, 1996.

[80] S. Van Huffel and J. Vandewalle. *The Total Least Squares Problem, Computational Aspects and Analysis.* SIAM, Philadelphia, PA, 1991.

[81] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data.* Prentice-Hall, Inc, Englewood Cliffs, NJ, 1988.

[82] T. Joachims. Text categorization with support vector machines. Technical Report LS VII Number 23, University of Dortmund, 1997. ftp://ftp-ai.infomatik.uni-dortmund.de/pub/Reports/report23.ps.Z.

[83] G. H. John, R. Kohavi, and K. Pfleger. Irrelevant features and the subset selection problem. In *Proceedings of the 11th International Conference on Machine Learning*, pages 121–129, San Mateo, CA, 1994. Morgan Kaufmann.

[84] E. L. Kaplan and P. Meier. Nonparametric estimation from incomplete observations. *Journal of the American Statistical Association*, 53:457–481, 1958.

[85] K. Kira and L. Rendell. The feature selection problem: Traditional methods and a new algorithm. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 129–134, San Mateo, CA, 1992. Morgan Kaufmann.

[86] David G. Kleinbaum. *Survival Analysis.* Springer-Verlag, New York, 1996.

[87] T. Kohonen. Self-organized formation of typologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.

[88] K. Koller and M. Sahami. Toward optimal feature selection. In *Machine Learning: Proceedings of the Thirteenth International Conference*, San Mateo, CA, 1996. Morgan Kaufmann.

[89] S. Kullback and R. A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22:76–86, 1951.

[90] P. A. Lahenbruch and R. M. Mickey. Estimation of error rates in discriminant analysis. *Technometrics*, 10:1–11, 1968.

[91] P. Langley, H. A. Simon, and G. L. Bradshaw. Heuristics for empirical discovery. In *Computational Models of Learning*, Berlin, 1987. Springer-Verlag.

[92] Y. le Cun, J. S. Denker, and S. A. Solla. Optimal brain damage. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems II (Denver 1989)*, pages 598–605, San Mateo, California, 1990. Morgan Kaufmann.

[93] D. B. Lenat. The ubiquity of discovery. *Artificial Intelligence*, 9:257–285, 1977.

[94] Z.-Q. Luo, J.-S. Pang, D. Ralph, and S.-Q. Wu. Mathematical programs with equilibrium constraints. *Mathematical Programming*, 75:19–76, 1996.

[95] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. Symp. Math. Statist. and Probability, 5th, Berkeley*, volume 1, pages 281–297, Berkeley, CA, 1967. Univ. of California Press. AD 66981.

[96] O. L. Mangasarian. Linear and nonlinear separation of patterns by linear programming. *Operations Research*, 13:444–452, 1965.

[97] O. L. Mangasarian. Multi-surface method of pattern separation. *IEEE Transactions on Information Theory*, IT-14:801–807, 1968.

[98] O. L. Mangasarian. *Nonlinear Programming.* McGraw–Hill, New York, 1969. Reprint: SIAM Classic in Applied Mathematics 10, 1994, Philadelphia.

[99] O. L. Mangasarian. Characterization of linear complementarity problems as linear programs. *Mathematical Programming Study*, 7:74–87, 1978.

[100] O. L. Mangasarian. Mathematical programming in neural networks. *ORSA Journal on Computing*, 5(4):349–360, 1993.

[101] O. L. Mangasarian. Misclassification minimization. *Journal of Global Optimization*, 5:309–323, 1994.

[102] O. L. Mangasarian. The linear complementarity problem as a separable bilinear program. *Journal of Global Optimization*, 6:153–161, 1995.

[103] O. L. Mangasarian. Machine learning via polyhedral concave minimization. In H. Fischer, B. Riedmueller, and S. Schaeffler, editors, *Applied Mathematics and Parallel Computing - Festschrift for Klaus Ritter*, pages 175–188. Physica-Verlag A Springer-Verlag Company, Heidelberg, 1996. ftp://ftp.cs.wisc.edu/math-prog/tech-reports/95-20.ps.Z.

[104] O. L. Mangasarian. Arbitrary-norm separating plane. Technical Report 97-07, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin,

May 1997. *Operations Research Letters*, submitted. ftp://ftp.cs.wisc.edu/math-prog/tech-reports/97-07.ps.Z.

[105] O. L. Mangasarian. Mathematical programming in data mining. *Data Mining and Knowledge Discovery*, 1(2):183–201, 1997. ftp://ftp.cs.wisc.edu/math-prog/tech-reports/96-05.ps.Z.

[106] O. L. Mangasarian. Minimum-support solutions of polyhedral concave programs. Technical Report 97-05, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, April 1997. Optimization, to appear. ftp://ftp.cs.wisc.edu/math-prog/tech-reports/97-05.ps.Z.

[107] O. L. Mangasarian. Solution of general linear complementarity problems via nondifferentiable concave minimization. *Acta Mathematica Vietnamica*, 22(1):199–205, 1997. ftp://ftp.cs.wisc.edu/math-prog/tech-reports/96-10.ps.Z.

[108] O. L. Mangasarian and R. R. Meyer. Nonlinear perturbation of linear programs. *SIAM Journal on Control and Optimization*, 17(6):745–752, November 1979.

[109] O. L. Mangasarian, W. N. Street, and W. H. Wolberg. Breast cancer diagnosis and prognosis via linear programming. *Operations Research*, 43(4):570–577, July-August 1995.

[110] MATLAB. *User's Guide.* The MathWorks, Inc., 1992.

[111] J. L. McClelland and D. E. Rummelhart. *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises.* MIT Press, Cambridge, Massachusetts, 1987.

[112] M. Meila and D. Heckerman. An experimental comparison of several clustering and initialization methods. Technical Report MSR-TR-98-06, Microsoft Research, Redmond, WA, 1998.

[113] G. Melli. Synthetic classification data sets (scds). http://fas.sfu.ca/cs/people/GradStudents/melli/SCDS/., 1997.

[114] R. R. Meyer. Private communication, August 24, 1998.

[115] R. R. Meyer. Private communication, August 25, 1998.

[116] Minitab, Inc. *Minitab Reference Manual, Release 9.* Minitab, Inc., College Station, PA, 1992.

[117] M. C. Mozer and P. Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems I*, pages 107–115, San Mateo, CA, 1989. Morgan Kaufmann.

[118] A. N. Mucciardi and E. E. Gose. A comparison of seven techniques for choosing subsets of pattern recognition properties. *IEEE Trans. Comput.*, C-20:1023–1031, September 1971.

[119] P. M. Murphy and D. W. Aha. UCI repository of machine learning databases. Technical report, Department of Information and Computer Science, University of California, Irvine, 1992. www.ics.uci.edu/˜mlearn/MLRepository.html.

[120] B. A. Murtagh and M. A. Saunders. MINOS 5.0 user's guide. Technical Report

SOL 83.20, Stanford University, December 1983. MINOS 5.4 Release Notes, December 1992.

[121] S. Murthy, S. Kasif, S. Salzberg, and R. Beigel. OC1: Randomized induction of oblique decision trees. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 322–327, Cambridge, MA 02142, 1993. The AAAI Press/The MIT Press.

[122] K. G. Murty. *Linear Programming.* John Wiley & Sons, New York, 1983.

[123] P. M. Narendra and K. Fukunaga. A branch and bound algorithm for feature subset selection. *IEEE Trans. Comput.*, C-26(9):917–922, September 1977.

[124] R. M. Neal and G. E. Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. Technical report, Dept. of Statistics and Dept. of Computer Science, University of Toronto, Toronto, Ontario, Canada, 1997.

[125] B. Noble and J. W. Daniel. *Applied Linear Algebra.* Prentice Hall, Englewood Cliffs, New Jersey, third edition, 1988.

[126] S. Odewahn, E. Stockwell, R. Pennington, R. Hummphreys, and W. Zumach. Automated star/galaxy discrimination with neural networks. *Astronomical Journal*, 103(1):318–331, 1992.

[127] E. Osuna, R. Freund, and F. Girosi. Training support vector machines: An application to face detection. In *IEEE Confernece on Computer Vision and Pattern Recognition, Puerto Rico, June 1997, 130-136*, 1997. http://www.ai.mit.edu/people/girosi/home-page/svm.html.

[128] M. L. Overton. A quadratically convergent method for minimizing a sum of Euclidean norms. *Mathematical Programming*, 27:34–63, 1983.

[129] G. Pagallo. Learning DNF by decision trees. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 639–644, 1989.

[130] G. Piatetsky-Shapiro, R. Brachman, T. Khabaza, W. Kloesgen, and E. Simoudis. An overview of issues in developing industrial data mining and knowledge discovery application. In *Proceedings, Second International Conference on Knowledge Discovery and Data Mining*. AAAI Press, 1996.

[131] J. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical Report 98-14, Microsoft Research, Redmond, Washington, April 1998. http://www.research.microsoft.com/~jplatt/smo.html.

[132] B. T. Polyak. *Introduction to Optimization.* Optimization Software, Inc., Publications Division, New York, 1987.

[133] J. R. Quinlan. Learning efficient classification procedures and their application to chess endgames. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 463–482, Palo Alto, CA, 1983. Tioga Publishing Company.

[134] J. R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA, 1993.

[135] M. R. Rao. Cluster analysis and mathematical programming. *Journal of the American Statistical Association*, 66:622–626, 1971.

[136] J. Rissanen. Stochastic complexity and modelling. *Ann. Statist.*, 14:1080–1100, 1986.

[137] R. T. Rockafellar. *Convex Analysis*. Princeton University Press, Princeton, New Jersey, 1970.

[138] J. B. Rosen, H. Park, and J. Glick. Total least norm formulation and solution for structured problems. *SIAM Journal on Matrix Analysis*, 17(1):110–128, January 1996.

[139] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing*, pages 318–362, Cambridge, Massachusetts, 1986. MIT Press.

[140] D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing*. MIT Press, Cambridge, Massachusetts, 1986.

[141] D. E. Rummelhard and D. Zipser. Feature discovery by competitive learning. *Congnitive Science*, 9:75–112, 1985.

[142] C. Schaffer. Overfitting avoidance as bias. *Machine Learning*, 10:153–178, 1993.

[143] B. Schölkopf. *Support Vector Learning*. PhD thesis, Technischen Universität Berlin, Berlin, 1997.

[144] B. Schölkopf, C. Burges, and V. Vapnik. Extracting support data for a given task. In *Proceedings of the First International Conference in Knowledge Disovery and Data Mining*, Menlo Park, CA, 1995. AAAI Press.

[145] B. Schölkopf, C. Burges, and V. Vapnik. Incorporating invariances in support vector learning machines. In *Artificial Neural Networks – ICANN96*, pages 47–52, Berlin, 1996. Springer Lecture Notes in Computer Science, Vol. 1112.

[146] B. Schölkopf, A. Smola, and K. R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 1998.

[147] S. Z. Selim and M. A. Ismail. K-Means-Type algorithms: a generalized convergence theorem and characterization of local optimality. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6:81–87, 1984.

[148] J. W. Shavlik and T. G. Dietterich (editors). *Readings in Machine Learning.* Morgan Kaufman, San Mateo, California, 1990.

[149] A. Smola. Regression estimation with support vector learning machines. Master's thesis, Technische Universität München, 1996.

[150] H. Späth. *Mathematical Algorithms for Linear Regression.* Academic Press, San Diego, 1992.

[151] P. Stolorz and R. Musick. Editorial. *Data Mining and Knowledge Discovery*, 1(4):339–341, 1997.

[152] M. Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society*, 36:111–147, 1974.

[153] G. Strang. *Introduction to Linear Algebra.* Wellesley-Cambridge Press, Wellesley, MA, 1993.

[154] W. N. Street. Cancer diagnosis and prognosis via linear-programming-based machine learning. Computer Sciences Department, Mathematical Programming Technical Report 94-14, University of Wisconsin, Madison, Wisconsin, August 1994.

[155] C. W. Therrien. *Discrete Random Signals and Statistical Signal Processing.* Prentice-Hall, Inc, Englewood Cliffs, NJ, 1992.

[156] V. Vapnik. Inductive principles of statistics and learning theory. In *Mathematical Perspectives on Neural Networks*, Mahwah, NJ, 1995. Lawrence Erlbaum.

[157] V. Vapnik and A. Chervonenkis. Uniform convergence of frequencies of occurence of events to their probabilities. *Dokl. Akad. Nauk SSSR*, 181:915–918, 1968.

[158] V. Vapnik and A. Chervonenkis. *Theory of Pattern Recognition [in Russian].* Nauka, Moscow, 1974. (German Translation: W. Wapnik & A. Tscherwonenkis, *Theorie der Zeichenerkennung*, Akademie-Verlag, Berlin, 1979).

[159] V. Vapnik, S. E. Golowich, and A. Smola. Support vector method for function approximation, regression estimation, and signal processing. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, Cambridge, MA, 1997. MIT Press.

[160] V. N. Vapnik. *Estimation of Dependences Based on Empirical Data.* Springer, New York, 1982.

[161] V. N. Vapnik. *The Nature of Statistical Learning Theory.* Springer, New York, 1995.

[162] G. Wahba. *Spline Models for Observational Data.* SIAM, Philadelphia, 1990.

[163] G. Wahba. RBF's, SBF's, TreeBF's, smoothing spline ANOVA: Representers and pseudo-representers for a dictionary of basis functions for penalized likelihood estimates. Talk given at the NIPS*96 Model Complexity Workshop, NIPS*96, Snowmass, CO, see ftp://ftp.stat.wisc.edu/pub/wahba/talks/nips.96/m-c.talk.ps, December 1996.

[164] G. Wahba. Support vector machines, reproducing kernel Hilbert spaces and the randomized GACV. Technical report no. 984, Department of Statistics, University of Wisconsin, Madison, WI 53706, 1997. ftp://ftp.stat.wisc.edu/pub/wahba/index.html.

[165] G. Wahba, Y. Wang, C. Gu, R. Klein, and B. Klein. Structured machine learning for 'soft' classification with smoothing spline anova and stacked tuning, testing and evaluation. In *Advances in Neural Information Processing Systems 6*, pages 415–422, San Mateo, California, 1994. Morgan Kaufmann.

[166] C. Wallace and P. Freeman. Estimation and inference by compact coding. *Journal of the Royal Statistical Society (B)*, 49:240–265, 1987.

[167] D. H. Wolpert, editor. *The Mathematics of Generalization*, Reading, MA, 1995. Addison-Wesley.

[168] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: A new data clustering algorithm and its applications. *Data Mining and Knowledge Discovery*, 1(2):141–182, 1997.