Penalised spline support vector classifiers: computational issues

BY J.T. ORMEROD, M.P. WAND AND INGE KOCH

School of Mathematics and Statistics,
University of New South Wales, Sydney 2052, Australia

14th May, 2007

ABSTRACT

We study computational issues for support vector classification with penalised spline kernels. We show that, compared with traditional kernels, computational times can be drastically reduced in large problems making such problems feasible for sample sizes as large as $\sim 10^6$. The optimisation technology known as *interior point methods* plays a central role. Penalised spline kernels are also shown to allow simple incorporation of low-dimensional structure such as additivity. This can aid both interpretability and performance.

Some keywords: Additive Models; Interior Point Methods; Low-dimensional Structure; Low-rank Kernels; Semiparametric Regression; Support Vector Machines.

1 Introduction

Support vector classifiers (SVC) are a relatively new family of classifiers that are enjoying increasing use and success and, according to some accounts (e.g. Breiman, 2001), are superseding neural network classifiers. Expositions of support support vector classification include Burges (1998), Cristianini & Shawe-Taylor (2000) and Schölkopf and Smola (2002). Different members of the family of support vector classifiers are distinguished by their *kernel*, a positive definite symmetric function on $\mathbb{R}^d \times \mathbb{R}^d$ where d is the dimension of the predictor space, and choice of a few parameters such as the scale of the kernel. A drawback of many of the commonly used kernels is that fitting algorithms are at least $O(n^2)$ where n is the size of the training set (Simon, 2004; Hush, Kelly, Scovel & Steinwart, 2006). This can hinder their application to large problems, although remedies based on approximation ideas have been proposed by, for example, Smola & Schölkopf (2000), Williams & Seeger (2001) and Schölkopf & Smola (2002, Chapter 10).

Recently, Pearce & Wand (2006) showed how the design structure of low-rank semiparametric regression models (e.g. Ruppert, Wand & Carroll, 2003) can be used in the support vector classification context. The basic ingredients are kernels arising from penalised splines. Such kernels have the following advantages:

- They are *low-rank* in the sense that their eigen-decomposition involves only K nonzero eigenvalues where K is the number of spline basis functions and is typically much smaller than n. An alternative way of describing the low-rank property is

that the Gram matrix factorises into the product of an $n \times K$ matrix and its transpose. The low-rank property lends itself to the use of *interior point methods* (Fine & Scheinberg, 2001; Schölkopf & Smola, 2002 and Ferris and Munson, 2003). While interior point methods can be used for any choice of kernel, in the case of low-rank kernel representations the cost of each iteration is accelerated from $O(n^3)$ to $O(nK^2)$ operations. This can be a drastic improvement upon common support vector classifiers making problems with n even in the hundreds of thousands feasible. Furthermore, optimality conditions for interior point algorithms are much more closely satisfied by interior point methods than decomposition type algorithms such as SMO (Schölkopf & Smola, 2002, Chapter 10). Implementation of these algorithms for penalised spline kernels is the central focus of this paper.

- The incorporation of low-dimensional structure such as additivity is relatively straightforward (Hastie & Tibshirani, 1990). Hastie, Tibshirani and Friedman (2001; Sections 2.5 and 12.3.4) demonstrate that classifiers that allow for low-dimensional structure can perform better than those that do not. Classifiers with low-dimensional structure are also more interpretable.

- They correspond to a finite-dimensional kernelisation of the original feature space. This permits easier software management. Further details on this aspect are given in Section 2.

A possible disadvantage of low-rank kernels is that the set of basis functions is finite and may not be as flexible as a full-rank kernel. However, several studies on low-rank splines and kernels (e.g. Schoenberg, 1968; Wahba, 1990; Hastie, 1996; Fine & Scheinberg, 2001, Schölkopf & Smola (2002) and Wood, 2003) have shown that the difference between low-rank and full-rank performance is often minimal.

Some discussion on the choice of low-rank kernels is in order. Most of the work in this area is for spline smoothing, rather than general reproducing kernel methods, but the principles are the same. There are two general approaches to construction of low-rank splines. One is to start with a full-rank kernel and then derive low-rank approximations (e.g. Hastie, 1996; Smola & Schölkopf, 2000; Williams & Seeger, 2001; Schölkopf & Smola, 2002; Wood, 2003). The other is to simply devise a 'sensible' low-rank spline algorithm (e.g. Eilers & Marx, 1996; Nychka *et al.*, 1998; Ruppert *et al.*, 2003; Yau, Kohn & Wood, 2003). Each have their advantages and disadvantages, but the latter can have significant computational advantages and are more readily interpretable (as illustrated in Figure 1). Details are given in Section 2.

The main purpose of this article is to show how interior point methods can be used to facilitate fast fitting of penalised spline support vector classifiers. The resulting classifier has linear storage requirements and, storage aside, is able to train massive training samples in reasonable time. We also describe some novel classification models based on low-dimensional thin plate splines and additivity structures which can have interpretability advantages.

The next section gives an overview of penalised spline support vector classifiers. In Section 3 we describe their efficient computation via interior point methods. Section 4 makes some comparisons between penalised spline and common support vector classifiers in terms of computational time and predictive accuracy. Some concluding discussion is given in Section 5.

2 Penalised Spline Support Vector Classifiers

Denote the training data by $(\mathbf{x}_i, y_i)$, $1 \leq i \leq n$, were $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$. This corresponds to two-class classification. Multiclass problems can be handled by application of two-class classification to various class pairs (e.g. Hastie, Tibshirani and Friedman, 2001, Section 12.3.7). We seek classifiers $f : \mathbb{R}^d \to \mathbb{R}$ such that a new observation $\mathbf{x} \in \mathbb{R}^d$ is classified according to $\text{sign}\{f(\mathbf{x})\}$. Throughout we assume the 'classical' $n \gg d$ situation. The reverse situation, sometimes labelled 'high dimension, low sample size', has received considerable recent attention, particularly in computational genomics (e.g. Dudoit, Fridyland and Speed, 2002). However penalised spline kernels are more suited to classical classification problems.

Penalised spline classifiers may be based on various full-dimensional or low-dimensional 'models' for f. For illustratory purposes take $d = 5$ with $\mathbf{x} = (x_1, \ldots, x_5)$. Possible models for $f(x_1, x_2, x_3, x_4, x_5)$ are

(A)	$f(x_1, x_2, x_3, x_4, x_5)$	(general quinvariate function)
(B)	$f_1(x_1) + f_2(x_2) + f_3(x_3)$ $+f_4(x_4) + f_5(x_5)$	(additive function of all 5 variables)
(C)	$f_1(x_1) + f_3(x_3) + f_4(x_4)$	(additive function of only 3 variables)
(D)	$f_{12}(x_1, x_2) + f_{345}(x_3, x_4, x_5)$	(sum of bivariate and trivariate functions)
(E)	$f_1(x_1) + \beta_4\, x_4$	(additive function of 2 variables, but one linear)

Note that the models (C) and (E) correspond to the situation where some of the predictors are deemed to have negligible predictive power for classification. Such *parsimonious* models are important in certain applications (most notably, data mining) where identification of the driving factors behind a particular outcome is of intrinsic interest.

Commonly used kernels in support vector classification software correspond to the full model (A). Penalised spline kernels can be tailored to any such model. The kernel arises from the basis functions used to model f. There are several families of basis functions that can be used to construct penalised spline models (e.g. Ruppert *et al.*, 2003, Section 3.7). Here we will limit discussion to a class of radially symmetric basis functions based on *thin plate splines* (French, Kammann and Wand, 2001). Suppose that a d'-dimensional function is required where $1 \leq d' \leq d$ and let m be an integer such that $2m > d'$. Then, for $\mathbf{x}' \in \mathbb{R}^{d'}$, we consider models of the form

$$f_{md'}(\mathbf{x}') = \sum_{j=1}^{p'} \beta_j \phi_j(\mathbf{x}') + \sum_{k=1}^{K'} u_k \psi_k(\mathbf{x}')$$

where $\{\phi_j\}$ is the set of all $p' = \binom{d'+m-1}{d'}$ d'-dimensional polynomials in the components of $\mathbf{x}'$ with degree less than m and

$$\psi_k(\mathbf{x}') \equiv \psi_k(\mathbf{x}'; m, d', \boldsymbol{\kappa}) \equiv k^{th} \text{ entry of } [r_{md'}(\mathbf{x}' - \boldsymbol{\kappa}_i)]_{1 \leq i \leq K'} [r_{md'}(\boldsymbol{\kappa}_i - \boldsymbol{\kappa}_{i'})]^{-1/2}_{1 \leq i, i' \leq K'}.$$

Here

$$r_{md'}(\mathbf{x}') = \begin{cases} \|\mathbf{x}'\|^{2m-d'} & d' \text{ odd} \\ \|\mathbf{x}'\|^{2m-d'} \log \|\mathbf{x}'\| & d' \text{ even} \end{cases}$$

and $\boldsymbol{\kappa}_1, \ldots, \boldsymbol{\kappa}_{K'} \in \mathbb{R}^{d'}$ is a set of K' *knots*. Full-rank thin plate spline models use $K = n$ and $\boldsymbol{\kappa}_k = \mathbf{x}'_k$, $1 \leq k \leq n$ where the $\mathbf{x}'_k$ are the d'-variate sub-vectors of the $\mathbf{x}_k$ corresponding to $\mathbf{x}' \in \mathbb{R}^{d'}$. Throughout we use $\|\mathbf{v}\| = \sqrt{\mathbf{v}^\mathsf{T}\mathbf{v}}$ to denote the length of the vector $\mathbf{v}$.

One common approach to low-rank spline smoothing (e.g. Ruppert *et al.*, 2003) is to use $K \ll n$ knots and choose the $\boldsymbol{\kappa}_k$ to 'mimic' the $\mathbf{x}_i'$s. A simple strategy is to draw a random sample of size K from the $\mathbf{x}_i'$s. Alternatively, one can use deterministic rules that aim to somehow 'fill the space' of the x_i's. For one-dimensional fitting ($d' = 1$) taking $\kappa_k \simeq (k/K)$th sample quantile of the unique x_i''s achieves this aim. For higher dimensions *distance-design* algorithms such as those used by Nychka and Saltzman (1998) can be used. Let $\mathcal{D}$ be a subset of observed points x_i called design points and $\mathcal{C}$ be a subset of observed points x_i called candidate points with $\mathcal{D} \cap \mathcal{C} = \phi$. Then the coverage of $\mathcal{C}$ by points in $\mathcal{D}$ is given by

$$C_{a,b}(\mathcal{D}) = \left(\sum_{x \in \mathcal{C}} d_a(x, \mathcal{D})^b \right)^{(1/b)} \tag{1}$$

where

$$d_a(x, \mathcal{D}) = \left(\sum_{u \in \mathcal{D}} \|x - u\|^a \right)^{(1/a)} \tag{2}$$

and $a < 0$ and $b > 0$. Minimising $C_{a,b}$ fills the space around the data. Minimisation is conducted by making pairwise swaps of points in $\mathcal{D}$ with points in $\mathcal{C}$ until the coverage $C_{a,b}$ does not decrease. If we choose $\mathcal{D}$ to be our set of knots then this procedure requires at least $O(Kn^2)$ computations and $O(nK)$ storage. Note that as $a \to -\infty$ and $b \to \infty$ minimising $C_{a,b}$ converges to the minimax space filling designs discussed in Johnson, Moore and Ylvisaker (1990) and with $a \to -\infty$ and $b = 1$ converges to the criteria used by the *CLARA* and *PAM* algorithms of Kaufman and Rousseeuw (1990). Note that the implementation of CLARA lowers computational speed by examining subsamples of the data. By doing this CLARA achieves approximate clustering in $O(SK^2n)$ computations (assuming S subsamples of size $O(K)$).

For general penalised spline support vector classification the model for f dictates the set of spline basis functions which, in turn, dictates the kernel. In the $d = 5$ example with $m = 2$ thin plate splines, model (C) leads to

$$\begin{aligned} f_C(\mathbf{x}) &= \beta_0 + \beta_1 x_1 + \sum_{k=1}^{K_1} u_{1k} \psi_k(x_1; 2, 1, \kappa^1) + \beta_3 x_3 + \sum_{k=1}^{K_3} u_{3k} \psi_k(x_3; 2, 1, \kappa^3) \\ &\quad + \beta_4 x_4 + \sum_{k=1}^{K_4} u_{4k} \psi_k(x_4; 2, 1, \kappa^4) \end{aligned}$$

where $\kappa^j = (\kappa^j_1, \ldots, \kappa^j_{K_j})$ is a set of univariate knots corresponding to x_j ($j = 1, 3, 4$). The kernel for this model is

$$\begin{aligned} \mathcal{K}_C(\mathbf{s}, \mathbf{t}) &= 1 + s_1 t_1 + s_3 t_3 + s_4 t_4 + \sum_{k=1}^{K_1} \psi_k(s_1; 2, 1, \kappa^1) \psi_k(t_1; 2, 1, \kappa^1) \\ &\quad + \sum_{k=1}^{K_3} \psi_k(s_3; 2, 1, \kappa^3) \psi_k(t_3; 2, 1, \kappa^3) + \sum_{k=1}^{K_4} \psi_k(s_4; 2, 1, \kappa^4) \psi_k(t_4; 2, 1, \kappa^4) \end{aligned}$$

for $\mathbf{s} = (s_1, \ldots, s_5), \mathbf{t} = (t_1, \ldots, t_5) \in \mathbb{R}^5$.

Model (D) has spline basis representation

$$f_D(\mathbf{x}) = \beta_0 + \boldsymbol{\beta}_{12}^{\mathsf{T}} [x_1 \; x_2]^{\mathsf{T}} + \sum_{k=1}^{K_{12}} u_{12k} \psi_k(x_1, x_2; 2, 2, \boldsymbol{\kappa}^{12})$$

$$+\boldsymbol{\beta}_{345}^{\mathsf{T}}[x_3\ x_4\ x_5]^{\mathsf{T}} + \sum_{k=1}^{K_{345}} u_{345k}\psi_k(x_3, x_4, x_5; 2, 3, \boldsymbol{\kappa}^{345})$$

where $\boldsymbol{\kappa}^{12}$ denotes a set of knots in the (x_1, x_2) space and $\boldsymbol{\kappa}^{345}$ denotes a set of knots in the (x_3, x_4, x_5) space. The corresponding kernel is

$$\begin{aligned}\mathcal{K}_D(\mathbf{s}, \mathbf{t}) &= 1 + \mathbf{s}^{\mathsf{T}}\mathbf{t} + \sum_{k=1}^{K_{12}} \psi_k(s_1, s_2; 2, 2, \boldsymbol{\kappa}^{12})\psi_k(t_1, t_2; 2, 2, \boldsymbol{\kappa}^{12}) \\ &\quad + \sum_{k=1}^{K_{345}} \psi_k(s_3, s_4, s_5; 2, 3, \boldsymbol{\kappa}^{345})\psi_k(t_3, t_4, t_5; 2, 3, \boldsymbol{\kappa}^{345}).\end{aligned}$$

Once the model, or kernel, is decided upon then there are two more choices to be made for penalised spline support vector classifiers:

(1) the subset of basis functions that are unpenalised, and

(2) the number of distinct penalisation parameters and their allocation to the penalised basis functions.

In support vector classification it is usual to just leave the intercept β_0 unpenalised. In spline smoothing all of the polynomial terms are usually left unpenalised. We will use $\mathbf{X}$ for the design matrix of unpenalised terms and $\mathbf{Z}$ for the design matrix of penalised terms. The respective coefficients will be denoted by $\boldsymbol{\beta}$ and $\mathbf{u}$. The ith fitted value is then

$$f(\mathbf{x}_i) = (\mathbf{X}\boldsymbol{\beta} + \mathbf{Z}\mathbf{u})_i.$$

If only the intercept is penalised then $\mathbf{X}$ is a column of ones and $\boldsymbol{\beta} = \beta_0$. Let

$$\mathbf{Z}\mathbf{u} = \sum_{\ell=1}^{L} \mathbf{Z}_\ell \mathbf{u}_\ell$$

for some partition $\mathbf{Z}_1, \ldots, \mathbf{Z}_L$ of $\mathbf{Z}$ such that each $\mathbf{u}_\ell$ has its own penalty parameter λ_ℓ. The 'natural' choice for the $\mathbf{Z}_\ell$ is that for which each predictor variable has its own smoothing parameters. So for model (D) with only β_0 being unpenalised we would have $L = 2$,

$$\mathbf{Z}_1 = [x_{1i}\ x_{2i}\ \underset{1\le k\le K_{12}}{\psi_k(x_{1i}, x_{2i}; 2, 2, \kappa^{12})}]_{1\le i\le n}$$

and

$$\mathbf{Z}_2 = [x_{3i}\ x_{4i}\ x_{5i}\ \underset{1\le k\le K_{345}}{\psi_k(x_{3i}, x_{4i}, x_{5i}; 2, 3, \kappa^{345})}]_{1\le i\le n}.$$

The penalised spline support vector classifier minimises

$$\sum_{i=1}^{n}\{1 - y_i(\mathbf{X}\boldsymbol{\beta} + \mathbf{Z}\mathbf{u})_i\}_+ + \sum_{\ell=1}^{L} \lambda_\ell \|\mathbf{u}_\ell\|^2.$$

This is equivalent to the constrained optimisation problem

$$\begin{aligned}&\min_{\boldsymbol{\beta},\mathbf{u}} \textstyle\sum_{\ell=1}^{L} \lambda_\ell \|\mathbf{u}_\ell\|^2 + \sum_{i=1}^{n} \xi_i \\ &\text{subject to } \xi_i \ge 0,\ y_i(\mathbf{X}\boldsymbol{\beta} + \mathbf{Z}\mathbf{u})_i \ge 1 - \xi_i \text{ for all } 1 \le i \le n.\end{aligned} \tag{3}$$

This problem, in turn, leads to the quadratic programming problem

$$\begin{aligned} &\min_{\boldsymbol{\alpha}}(-\mathbf{1}^\mathsf{T}\boldsymbol{\alpha} + \tfrac{1}{2}\boldsymbol{\alpha}^\mathsf{T}\mathbf{D}\boldsymbol{\alpha}) \\ &\text{subject to } 0 \leq \alpha_i \leq 1, \text{ for all } 1 \leq i \leq n, \text{ and } \mathbf{X}^\mathsf{T}(\boldsymbol{\alpha} \odot \mathbf{y}) = \mathbf{0} \end{aligned} \tag{4}$$

where

$$\mathbf{D} = \tfrac{1}{2}(\mathbf{y}\mathbf{y}^\mathsf{T}) \odot (\mathbf{Z}\boldsymbol{\Lambda}^{-1}\mathbf{Z}^\mathsf{T}) \quad \text{and} \quad \boldsymbol{\Lambda} = \text{diag}(\lambda_1\mathbf{1}, \ldots, \lambda_L\mathbf{1})$$

Here $\mathbf{A} \odot \mathbf{B}$ denotes the element-wise product of equal-sized matrices $\mathbf{A}$ and $\mathbf{B}$ and $\mathbf{1}$ is the vector of ones of appropriate length. See Pearce & Wand (2006) for details. Since the Gram matrix admits the factorisation

$$\tfrac{1}{2}\mathbf{Z}\boldsymbol{\Lambda}^{-1}\mathbf{Z}^\mathsf{T} = \widetilde{\mathbf{Z}}\widetilde{\mathbf{Z}}^\mathsf{T} \quad \text{where} \quad \widetilde{\mathbf{Z}} = \mathbf{Z}(2\boldsymbol{\Lambda})^{-1/2}$$

the quadratic programming problem becomes

$$\begin{aligned} &\min_{\boldsymbol{\alpha}}[-\mathbf{1}^\mathsf{T}\boldsymbol{\alpha} + \tfrac{1}{2}\boldsymbol{\alpha}^\mathsf{T}\{(\mathbf{y}\mathbf{y}^\mathsf{T}) \odot (\widetilde{\mathbf{Z}}\widetilde{\mathbf{Z}}^\mathsf{T})\}\boldsymbol{\alpha}] \\ &\text{subject to } 0 \leq \alpha_i \leq 1, \text{ for all } 1 \leq i \leq n, \text{ and } \mathbf{X}^\mathsf{T}(\boldsymbol{\alpha} \odot \mathbf{y}) = \mathbf{0}. \end{aligned} \tag{5}$$

The 'bottom line' of this section is that penalised spline support vector classifiers are just ordinary hyperplane support vector classifiers with the original features $\mathbf{x}_i \in \mathbb{R}^d$ replaced by $\widetilde{\mathbf{z}}_i \in \mathbb{R}^K$, $1 \leq i \leq n$, corresponding to the rows of $\widetilde{\mathbf{Z}}$ (with K denoting the number of columns in $\widetilde{\mathbf{Z}}$). This makes software management relatively simple since only $\mathbf{y}$, $\mathbf{X}$ and the $n \times K$ matrix $\widetilde{\mathbf{Z}}$ need to be passed to a quadratic programming routine. Software for general support vector classifiers either needs to deal with $O(n^2)$ storage of the Gram matrix or evaluate the kernel inside an algorithm such as Sequential Minimal Optimisation (SMO) (e.g. Cristianini & Shawe-Taylor, 2000). An even bigger payoff is the fact that the Gram matrix $\widetilde{\mathbf{Z}}\widetilde{\mathbf{Z}}^\mathsf{T}$ has rank K. The next section summarises an efficient algorithm for solving the problem when this is the case.

3 Interior Point Methods

Interior Point Methods (IPM) are one of the most important developments in optimisation in the last two decades. In this section we provide the minimal information needed to code a reasonably efficient interior point method for support vector classification. More efficient methods exist, but they involve extra complexity which obscure the main ideas. Extensive literature exists on interior point methods. For an introduction the reader is referred to Wright (1997), Nocedal & Wright (1999) and Boyd & Vandenberghe (2004). For interior point methods in the context of support vector machines the reader is referred to Fine & Scheinberg (2001), Schölkopf & Smola, (2002), Ferris & Munson (2003) and Vandenberghe & Comanor (2003).

3.1 Description

Interior point methods have been developed to solve most convex programming problems (see Boyd & Vandenberghe, 2004). However, unless special structure is available, these methods are restrictive when the dimension of the optimisation problem becomes large.

Our goal is to solve the dual optimisation problem (5). Its corresponding primal problem may be written

$$\begin{aligned} &\min_{\boldsymbol{\beta},\mathbf{u},\boldsymbol{\xi},\boldsymbol{\zeta}} \|\mathbf{u}\|^2 + \textstyle\sum_{i=1}^n \xi_i \\ &\text{subject to } \xi_i, \zeta_i \geq 0,\ y_i(\mathbf{X}\boldsymbol{\beta} + \widetilde{\mathbf{Z}}\mathbf{u})_i + \xi_i - \zeta_i = 1, \quad \text{for all } 1 \leq i \leq n \end{aligned} \tag{6}$$

where $\boldsymbol{\xi} \equiv (\xi_1, \ldots, \xi_n)$ and $\boldsymbol{\zeta} \equiv (\zeta_1, \ldots, \zeta_n)$. Note that this problem corresponds to (3) but with $\mathbf{Z}$ replaced by $\widetilde{\mathbf{Z}}$ and the introduction of slack variables ζ_i, $1 \leq i \leq n$. For compactness of notation we define

$$\mathbf{A} \equiv \mathbf{X}^\mathsf{T}\text{diag}(\mathbf{y}), \quad \text{and} \quad \mathbf{V} \equiv \widetilde{\mathbf{Z}}^\mathsf{T}\text{diag}(\mathbf{y}).$$

IPMs start with an initial guess and iteratively find better solutions until some convergence criteria is reached. They focus on a system of quadratic equations made up of the primal constraints, dual constrains and the perturbed complementary slackness conditions induced by a log barrier function (see Boyd & Vandenberghe, 2004). These conditions are

$$\begin{aligned} \mathbf{V}^\mathsf{T}\mathbf{V}\boldsymbol{\alpha} + \mathbf{A}^\mathsf{T}\boldsymbol{\beta} + \boldsymbol{\xi} - \boldsymbol{\zeta} &= \mathbf{1} \quad \text{(Primal Feasibility)} \\ \mathbf{A}\boldsymbol{\alpha} &= \mathbf{1} \quad \text{(Dual Feasibility)} \\ (\alpha_i - 1)\xi_i &= t \quad \text{(Perturbed Complementary Slackness)} \\ \alpha_i\zeta_i &= t \quad \text{(Perturbed Complementary Slackness)} \end{aligned} \tag{7}$$

where t is some positive constant. Let the solution of such a system of equations be $\widehat{\boldsymbol{\alpha}}$, $\widehat{\boldsymbol{\beta}}$, $\widehat{\boldsymbol{\xi}}$ and $\widehat{\boldsymbol{\zeta}}$ and let P^* denote the optimal value of the primal objective (6). Then it can be shown (Boyd & Vandenberghe, 2004) that

$$\tfrac{1}{2}\widehat{\boldsymbol{\alpha}}^\mathsf{T}\mathbf{V}^\mathsf{T}\mathbf{V}\widehat{\boldsymbol{\alpha}} - \mathbf{1}^\mathsf{T}\widehat{\boldsymbol{\alpha}} - P^* \leq 2nt.$$

Hence reduction of t leads to solutions of the original and the perturbed problems becoming closer.

Let $\boldsymbol{\alpha}^{(j)}$, $\boldsymbol{\beta}^{(j)}$, $\boldsymbol{\zeta}^{(j)}$, $\boldsymbol{\xi}^{(j)}$ denote the values of $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{\zeta}$ and $\boldsymbol{\xi}$ after the jth iteration of the interior point method. A "cold" initial point is calculated using

$$\alpha_i^{(0)} = \varepsilon,\ \boldsymbol{\beta}^{(0)} = \mathbf{0},\ \xi_i^{(0)} = \max(\varepsilon, s_i) \quad \text{and} \quad \zeta_i^{(0)} = \max(\varepsilon, \xi_i^{(0)} - s_i)$$

where $s = \mathbf{1} - \mathbf{V}^\mathsf{T}\mathbf{V}\boldsymbol{\alpha} - \mathbf{A}^\mathsf{T}\boldsymbol{\beta}$ and ε is a small constant, say $\varepsilon = 10^{-2}$. This is a good starting point when the number of support vectors is small.

The system of equations (7) cannot be easily solved. Instead, in the spirit of Newton's method, we linearise around our current point by substituting

$$\boldsymbol{\alpha}^{(j)} + \Delta\boldsymbol{\alpha},\ \boldsymbol{\beta}^{(j)} + \Delta\boldsymbol{\beta}, \quad \boldsymbol{\xi}^{(j)} + \Delta\xi \quad \text{and} \quad \boldsymbol{\zeta}^{(j)} + \Delta\boldsymbol{\zeta}$$

into (7) and "solve" the resulting equations, in this order, to get the search direction vector $(\Delta\boldsymbol{\alpha}, \Delta\boldsymbol{\beta}, \Delta\boldsymbol{\xi}, \Delta\boldsymbol{\zeta})$:

$$\begin{aligned} \Delta\boldsymbol{\beta} &= \{\mathbf{A}^\mathsf{T}(\mathbf{V}^\mathsf{T}\mathbf{V} + \mathbf{D})^{-1}\mathbf{A}\}^{-1}(\mathbf{A}^\mathsf{T}\mathbf{r}_5 - \mathbf{r}_2), \\ \Delta\boldsymbol{\alpha} &= \mathbf{r}_5 - (\mathbf{V}^\mathsf{T}\mathbf{V} + \mathbf{D})^{-1}\mathbf{A}\Delta\boldsymbol{\beta}, \\ \Delta\zeta_i &= r_{3i} - \zeta_i^{(j)}\Delta\alpha_i/\alpha_i^{(j)}, \\ \Delta\xi_i &= r_{4i} + \xi_i^{(j)}\Delta\alpha_i/(1 - \alpha_i^{(j)}) \end{aligned} \tag{8}$$

where

$$\begin{aligned}
\mathbf{r}_1 &= \mathbf{1} - \mathbf{V}^\mathsf{T}\mathbf{V}\boldsymbol{\alpha}^{(j)} - \mathbf{A}^\mathsf{T}\boldsymbol{\beta}^{(j)} - \xi^{(j)} + \zeta^{(j)}, \\
\mathbf{r}_2 &= -\mathbf{A}\boldsymbol{\alpha}^{(j)}, \\
r_{3i} &= (t - \Delta\alpha_i\Delta\zeta_i)/\alpha_i^{(j)} - \zeta_i^{(j)}, \\
r_{4i} &= (t + \Delta\alpha_i\Delta\xi_i)/(1 - \alpha_i^{(j)}) - \xi_i^{(j)}, \\
\mathbf{r}_5 &= (\mathbf{V}^\mathsf{T}\mathbf{V} + \mathbf{D})^{-1}(\mathbf{r}_1 + \mathbf{r}_3 - \mathbf{r}_4)
\end{aligned} \tag{9}$$

and

$$\mathbf{D} = \operatorname*{diag}_{1\le i\le n}\{\xi^{(j)}/(1 - \alpha_i^{(j)}) + \zeta_i^{(j)}/\alpha_i^{(j)}\}.$$

Newton's method substitutes

$$\Delta\boldsymbol{\alpha} = \Delta\boldsymbol{\beta} = \Delta\boldsymbol{\xi} = \Delta\boldsymbol{\zeta} = \mathbf{0}$$

into (9) for a given value of t and then uses the values in (9) to calculate (8). However a similar method, the predictor-corrector method, usually accelerates the convergence of the algorithm (Mehrotra, 1992). In order to calculate the Newton predictor-corrector direction $\Delta\boldsymbol{\alpha}$, $\Delta\boldsymbol{\beta}$, $\Delta\xi$ and $\Delta\zeta$ we

1. find $\mathbf{r}_1, \ldots, \mathbf{r}_4$ by substituting $\boldsymbol{\alpha}^{(j)}$, $\boldsymbol{\beta}^{(j)}$, $\xi^{(j)}$, $\zeta^{(j)}$, $t = 0$, $\Delta\boldsymbol{\alpha} = 0$, $\Delta\boldsymbol{\beta} = 0$, $\Delta\xi = 0$ and $\Delta\zeta = 0$ into (9) and then calculate (8); and
2. recalculate $\mathbf{r}_3$ and $\mathbf{r}_4$ by substituting $\boldsymbol{\alpha}^{(j)}$, $\boldsymbol{\beta}^{(j)}$, $\xi^{(j)}$, $\zeta^{(j)}$ and the values of $\Delta\boldsymbol{\alpha}$, $\Delta\boldsymbol{\beta}$, $\Delta\xi$ and $\Delta\zeta$ from Step 1 into (9) and then recalculate (8).

Once we have a search direction we take the maximum step size $\tau \in (0, 1)$ such that $0 \le \boldsymbol{\alpha}^{(j)} + \tau\Delta\boldsymbol{\alpha} \le 1$, $\xi^{(j)} + \tau\Delta\xi \ge 0$ and $\zeta^{(j)} + \tau\Delta\zeta \ge 0$. This method of finding the step size is called simple dampening. Other step lengths exist (see Mészáros, 1999). Once the step size τ is found we update our values using

$$\begin{aligned}
\boldsymbol{\alpha}^{(j+1)} &= \boldsymbol{\alpha}^{(j)} + (1 - \varepsilon)\tau\Delta\boldsymbol{\alpha} \\
\boldsymbol{\beta}^{(j+1)} &= \boldsymbol{\beta}^{(j)} + (1 - \varepsilon)\tau\Delta\boldsymbol{\beta} \\
\xi^{(j+1)} &= \boldsymbol{\xi}^{(j)} + (1 - \varepsilon)\tau\Delta\boldsymbol{\xi} \\
\zeta^{(j+1)} &= \boldsymbol{\zeta}^{(j)} + (1 - \varepsilon)\tau\Delta\boldsymbol{\zeta}
\end{aligned} \tag{10}$$

The factor $(1 - \varepsilon)$ is included to ensure numerical feasibility. At each iteration we reduce t using

$$t = \frac{(\boldsymbol{\alpha}^{(j)T}\zeta^{(j)} + (1 - \boldsymbol{\alpha}^{(j)})^\mathsf{T}\xi^{(i)})(1 - \tau + \varepsilon)}{n(10 + \tau)^2}. \tag{11}$$

We stop when

$$\frac{\boldsymbol{\alpha}^{(j)T}\zeta^{(j)} + (1 - \boldsymbol{\alpha}^{(j)})^\mathsf{T}\xi^{(j)}}{\frac{1}{2}\boldsymbol{\alpha}^{(j)T}\mathbf{V}^\mathsf{T}\mathbf{V}\boldsymbol{\alpha}^{(j)} + \mathbf{1}^\mathsf{T}\boldsymbol{\alpha}^{(j)} + \mathbf{1}^\mathsf{T}\xi^{(j)}} \le \delta \tag{12}$$

for some tolerance $\delta > 0$.

3.2 Iteration Cost

For support vector machines it is common to have $K, d \ll n$. All steps are less than cubic in the number of operations so we can effectively ignore costs that do not involve n.

The main cost in IPMs for support vector classification is solving systems of the form $(\mathbf{V}^\mathsf{T}\mathbf{V}\boldsymbol{a} + \mathbf{D}) = \mathbf{b}$. Forming $\mathbf{V}^\mathsf{T}\mathbf{V}$ explicitly is expensive both computationally and in terms of storage. If we form $\mathbf{V}^\mathsf{T}\mathbf{V}$ explicitly then factorising $\mathbf{V}^\mathsf{T}\mathbf{V} + \mathbf{D}$ requires $O(n^3)$

operations and $O(n^2)$ storage. Much cheaper alternatives include use of the Sherman-Morrison-Woodbury formula and product form Cholesky factorisation. Each require $O(nK^2)$ operations and $O(nK)$ storage. The Sherman-Morrison-Woodbury formula is

$$(\mathbf{V}^\mathsf{T}\mathbf{V} + \mathbf{D})^{-1} = \mathbf{D}^{-1} - \mathbf{D}^{-1}\mathbf{V}^\mathsf{T}(\mathbf{I} + \mathbf{V}\mathbf{D}^{-1}\mathbf{V}^\mathsf{T})^{-1}\mathbf{V}\mathbf{D}^{-1}. \tag{13}$$

Note that $\mathbf{I} + \mathbf{V}\mathbf{D}^{-1}\mathbf{V}^\mathsf{T}$ is generally positive definite and can be factorised efficiently using Cholesky factorisation in $O(K^3)$ operations. However the main cost in calculating (13) is calculating $\mathbf{V}\mathbf{D}^{-1}\mathbf{V}^\mathsf{T}$ which requires $O(nK^2)$ operations. Product form Cholesky factorisation is more numerically stable but its description is more involved. Details on this approach are given in Fine and Scheinberg (2001).

3.3 Number of iterations

The overall complexity of the algorithm greatly depends on the number of iterations before convergence criteria are satisfied. The number of iterations depends on the choice of starting point, the method used to find the search direction, the step-size used to find the next iterate and how the parameter t is reduced. It can be shown that a naïvely coded IPM with a Newton predictor-corrector step direction gives a theoretical bound of $O(n)$ iterations for convergence (Boyd & Vandenberghe, 2004). State-of-the-art algorithms have been shown to have a worst case complexity of $O(\sqrt{n})$. However, extensive numerical experience shows that the number of iterations for IPMs is *almost constant*. See, for example, Fine & Scheinberg (2001).

4 Comparisons

Most of this section deals with additive functions of all variables as described in model (B). For these models we present some time comparisons based on different implementations. We compare the misclassification rates arising from different kernels for a number of well-known data sets. Lastly we present some preliminary results on extensions to bivariate models.

4.1 Kernels and Settings

We compare the performance of three different kernels. These are

1. the linear kernel (referred to as *linear*)

$$\mathcal{K}(\mathbf{s}, \mathbf{t}) = \mathbf{s}^T\mathbf{t};$$

2. the radial basis function kernel (referred to as *RBF*)

$$\mathcal{K}(\mathbf{s}, \mathbf{t}) = \exp(-\gamma\|\mathbf{s} - \mathbf{t}\|^2)$$

   for some $\gamma > 0$; and

3. the truncated lines penalised spline kernel (referred to as *PSVC*)

$$\mathcal{K}(\mathbf{s}, \mathbf{t}) = \sum_{j=1}^{d}\sum_{k=1}^{K_j}(s_j - \kappa_{jk})_+(t_j - \kappa_{jk})_+,$$

   where, for $1 \le k \le K_j$, κ_{jk} is the kth knot for the jth predictor.

Note that the linear and truncated lines penalised spline kernel can be seen as different cases of model (B) whereas the RBF kernel is an example of models of type (A). Having decided upon the kernels, we make the following choices for the subset of basis functions and penalisation parameters.

- $K_j = 20$ knots for each predictor, with κ_{jk} equal to the $(\frac{k+1}{K_j+2})$th sample quantile of the unique predictor values.
- Linear and intercept terms are unpenalised for the penalised spline so that

$$\mathbf{X} = [1 \; x_{i1} \; \ldots \; x_{id}]_{1 \le i \le n}$$

- The intercept term is unpenalised for the radial basis function and linear kernels.
- We use $\gamma = 1/d$ as the default value in the `svm()` function of the `R` package `e1071`.
- Choices for the smoothing parameters λ are given separately in the following sections.

Note that the `svm()` function in `R` is restricted to the case where only the intercept term is unpenalised. For the more general case one may need to resort to standard convex quadratic programming software such as that provided by the `quadprog` package in `R`.

In the computations described in the next sections, we use the standardised or scaled data.

4.2 Timing Comparisons

We compare the computation times of two different `MATLAB` implementations of the interior point method described in Section 3 with the `R` package `quadprog` (Turlach & Weingessel, 2006) which has a `Fortran` back-end for penalised spline support vector classifiers using the PSVC kernel described in Section 4.1. The two different `MATLAB` implementations differ in the method used to 'invert' $\mathbf{V}^\mathsf{T}\mathbf{V} + \mathbf{D}$. The first implementation 'inverts' $\mathbf{V}^\mathsf{T}\mathbf{V} + \mathbf{D}$ using `MATLAB`'s inbuilt $\backslash$ (backslash) operator. We refer to this method as *FULL*. The second implementation 'inverts' $\mathbf{V}^\mathsf{T}\mathbf{V} + \mathbf{D}$ using Sherman-Morrison-Woodbury's formula (13). We refer to this implementation as *SMW*. The `MATLAB` implementations terminate when the duality gap is smaller than 10^{-8}. The `R` package `quadprog` is an active-set method and so cannot be compared in terms of inverting the $\mathbf{V}^\mathsf{T}\mathbf{V} + \mathbf{D}$ matrix. The `quadprog` package stops when no constraints are violated in the active set.

We base the time comparisons on the 4 dimensional 'skin of the orange' setting described in Section 12.3.4 of Hastie, Tibshirani & Friedman (2001). The 'skin of the orange' data is generated by simulating two classes of points. Each data point from the first class is simulated from 4 independent standard normally distributed random variables X_1, X_2, X_3, X_4 while each data point from the second class is simulated from 4 independent standard normally distributed random variables X_1, X_2, X_3, X_4 conditioned on $9 \le \sum_{i=1}^{4} X_i^2 \le 16$.

Of particular interest is the effect of the sample size on computing times. For this reason we use sample sizes $n = 200, 1000$ and 5000. In all simulations other than the case of $n = 5000$ for `R`'s QP method, we used 50 runs. For the largest sample size with `R` the times are based on 10 runs only. The mean times and standard errors are given in Table 1 . We choose the smoothing parameters $\lambda_1, \ldots, \lambda_d$ for the penalised spline SVC so that each function has approximately 6 degrees of freedom.

	QP	FULL	SMW	$\frac{\text{QP}}{\text{SMW}}$	$\frac{\text{FULL}}{\text{SMW}}$
n=200	0.49 (0.0007)	0.53 (0.0004)	0.05 (0.0007)	9.80	10.06
n=1,000	68.51 (0.1401)	4.95 (0.0145)	0.50 (0.0016)	137.02	9.90
n=5,000	9210.57 (4.9080)	511.76 (0.1773)	3.98 (0.0027)	2314.21	128.58

Table 1: *Average times in seconds for the 4 dimensional 'skin of the orange' example using the two MATLAB implementations of the interior point method FULL and SMW and direct quadratic programming (QP) over 50 trials. Standard errors are given in brackets.*

The computations were performed on dual Opteron 2.0 GHz CPUs with 4 GB RAM and MATLAB version 7.01 and R version 2.0.0. In addition to the mean times in seconds and their standard errors, we also show ratios of times with the SMW implementations in the last two columns of Table 1 . The ratios provide further insight, because they are less dependent on changes in the computing environment. Nevertheless, the average times themselves give a real life aspect to the problem in that they indicate how long a user would have to wait for classifications in 2007 using a typical computing environment.

The comparisons in Table 1 show that huge time and storage savings can be obtained from using custom built convex quadratic programming solvers for low-rank kernels. In particular it has been possible to solve support vector classification problems with more than 10^6 training points, a task which would be practically impossible for general convex quadratic programming problems.

4.3 Performance Comparisons

We return to the three kernels listed in Section 4.1 and compare their performance for a number of well known real data sets which are available on the UCI Machine Learning Repository (Blake & Merz, 1998). In addition we have included the 2-dimensional *Checker* data which can be obtained on-line (Ho and Kleinberg, 1996), and the *Skin* ('skin of the orange') data sets of Hastie *et al*. (2001).

As all data sets have labels, classification *performance* of the three kernels is measured in terms of the misclassification rate. In our calculations we choose the smoothing (or 'cost') parameter for the linear and radial basis SVCs via 10-fold cross-validation using $\lambda_i = \lambda$ for all i and λ being chosen from 50 logarithmically equally spaced points between 2^{-15} and 2^{15}.

Our results are reported in Table 2 . We calculate the mean misclassification rate over 50 runs based on 10-fold cross-validation. We then determine the minimum over all 50 values of the smoothing parameter. This minimum value is our quoted misclassification rate. Standard errors at the minimum are given in brackets. The table lists the data sets together with their sample size and dimension or number of features, so the quantity d in the table refers to the dimension and does not count the labels as a dimension.

We also calculated average misclassification rates based on 50 runs with random 25% and 40% subsets of the data held back for testing. However these results were fairly similar to those given in Table 2 and so are not included.

Table 2 shows that the classification performance of PSVC, the additive penalised spline SVC, is comparable with (or better than) that of the RBF, the radial basis SVC, in all cases other than the *Checker* data set.

In addition, as previously stated, the use of the truncated lines penalised spline (and similar) kernels are inherently more interpretable. Figure 1 provides an illustration of a penalised spline support vector machine classification. The model is an additive function

Data	n	d	Linear	RBF	PSVC
Balance	625	4	4.76 (0.13)	1.75 (0.08)	0.63 (0.04)
Bupa	345	6	30.29 (0.50)	29.43 (0.46)	26.00 (0.40)
Checker	1000	2	48.60 (0.23)	3.10 (0.04)	39.10 (0.13)
Cmc	1473	10	31.44 (0.09)	28.95 (0.08)	27.26 (0.07)
Haberman	306	3	26.11 (0.50)	26.42 (0.49)	24.18 (0.43)
Pid	768	9	22.03 (0.17)	23.18 (0.19)	22.67 (0.19)
Skin200	200	4	44.50 (0.42)	4.50 (0.35)	5.00 (0.29)
Skin1000	1000	4	48.20 (0.30)	4.20 (0.05)	4.00 (0.05)
Votes	435	16	4.09 (0.18)	3.41 (0.09)	3.86 (0.17)
Wbcd	569	31	2.89 (0.08)	2.90 (0.07)	2.92 (0.04)

Table 2: *Average (standard error) misclassification rates based on 10-fold cross-validation using a Linear, RBF and PSVC.*

of 16 predictors of spam versus ordinary e-mail, with spam messages coded as $+1$ and ordinary messages coded as -1. See Hastie, Tibshirani and Friedman (2001) for a description of the these 'spam' data. Each panel shows the slice of the classification surface for the labelled predictor, with all other predictors set to their medians. Assuming the model in some way reflects reality, it appears, for example, that frequency of the word 'free' has a monotonic effect on classification while frequency of exclamation marks (`ch!`) has a non-monotonic effect. Thus increasing the word 'free' in an email increases the chance the email will be classified as spam whereas the chances increase or decrease depending on how many exclamation marks are already in the email. In some classification contexts, the type of relationship may be important for interpretation. Note that Figure 1 is visually similar to Figure 9.1 in Hastie, Tibshirani and Friedman (2001) with differences occurring mainly where data are sparse.

4.4 Extension to Bivariate Models

In the previous section we compared misclassification rates for a number of different kernels. In all cases (apart from the *Checker* example) PSVC performed at least as well as the other kernel methods and often better. For this reason we focus on penalised spline kernels and consider bivariate models such as

$$f_{p(1)p(2)}(x_{p(1)}, x_{p(2)}) + f_{p(3)p(4)}(x_{p(3)}, x_{p(4)}) + f_{p(5)p(6)}(x_{p(5)}, x_{p(6)}), \tag{14}$$

where $(p(1), \ldots, p(d))$ is a permutation of $(1, \ldots, d)$. This model is most similar to model (D), but is restricted to bivariate functions.

The design matrix consists of the constant term only, so that $\mathbf{X}$ is a vector of ones, and the $\mathbf{Z}$ matrix is extended to contain linear, mixed and quadratic terms. We consider three different forms for $\mathbf{Z}$:

- $\mathbf{Z}_1$ contains mixed terms $x_{p(j)}x_{p(j+1)}$ only;
- $\mathbf{Z}_2$ contains linear terms and mixed terms;
- $\mathbf{Z}_3$ contains linear, quadratic and mixed terms.

More specifically, we consider the entries in the $\mathbf{Z}$ matrix which arise from the linear, mixed and quadratic contributions. Let $\mathbf{Z}_{\text{lin}}$ denote the design matrix of the linear terms.

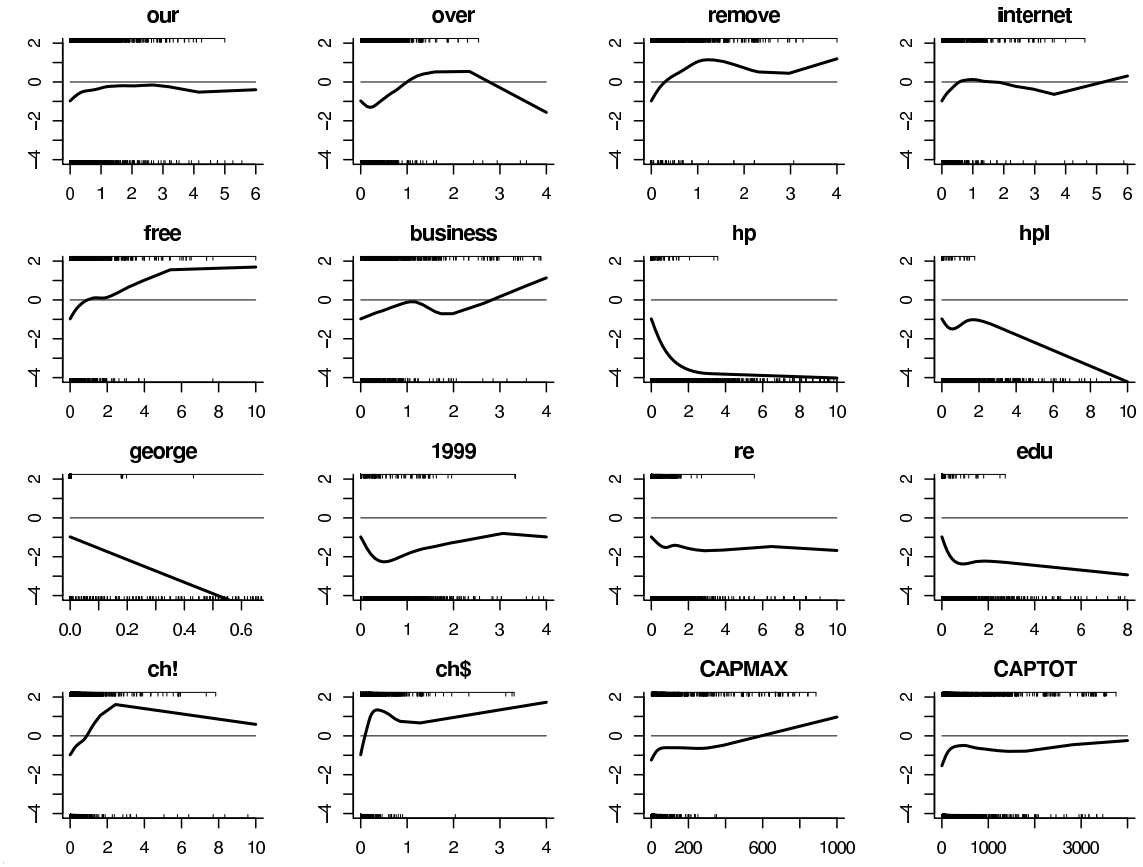

Figure 1: *A penalised spline support vector classifier for the 'spam' data. An additive model version is used. The tick-marks show the predictor values: spam messages along the top, normal messages along the bottom.*

The ith row of $\mathbf{Z}_{\text{lin}}$ consists of the terms

$$(x_{ij} - \kappa_{jk})_+ \quad \text{for } 1 \le j \le d,\ 1 \le k \le K_j. \tag{15}$$

Similarly let $\mathbf{Z}_{\text{quad}}$ denote the design matrix of the quadratic terms. The ith row of $\mathbf{Z}_{\text{quad}}$ consists of the terms

$$(x_{ij} - \kappa_{jk})_+^2 \quad \text{for } 1 \le j \le d,\ 1 \le k \le K_j. \tag{16}$$

Finally, terms of the form

$$(x_{ip(j)} - \kappa_{p(j)k})_+ \times (x_{ip(j+1)} - \kappa_{p(j+1)1})_+, \ldots, (x_{ip(j)} - \kappa_{p(j)k})_+ \times (x_{ip(j+1)} - \kappa_{p(j+1)K_{p(j+1)}})_+ \tag{17}$$

for $p(j) = 1, 3, \ldots d-1,\ 1 \le k \le K_{p(j)}$ contribute to the ith row of the design matrix $\mathbf{Z}_{\text{mix}}$ of the mixed terms. Here $p(j)$ and $p(j+1)$ denote consecutive terms of the permutation in (14). Using this notation, the three different $\mathbf{Z}$ matrices are

$$\begin{aligned} \mathbf{Z}_1 &= \mathbf{Z}_{\text{mix}} \\ \mathbf{Z}_2 &= [\mathbf{Z}_{\text{lin}}\ \mathbf{Z}_{\text{mix}}] \\ \mathbf{Z}_3 &= [\mathbf{Z}_{\text{lin}}\ \mathbf{Z}_{\text{quad}}\ \mathbf{Z}_{\text{mix}}]. \end{aligned} \tag{18}$$

In analogy with the one-dimensional results we calculate the best misclassification rate over a range of λ values (same as in the univariate case) via 10-fold cross-validation.

Data	n	d	1-d Results	2-d Results	Selected Model
Iris	150	4	3.33 (0.29)	1.33 (0.23)	$\mathbf{Z}_1$
Checker	1000	2	39.1 (0.13)	26.3 (0.18)	$\mathbf{Z}_3$
Bupa	345	6	26.0 (0.40)	23.7 (0.49)	$\mathbf{Z}_1$

Table 3: *Mean misclassification rates based on 10-fold cross-validation with bivariate PSVC. Standard errors are given in brackets.*

We apply these models to three different data sets: the well known *Iris* data set; the *Checker* data set; and the *Bupa liver* data set. We have not used the 4-dimensional *Balance* data set since the univariate PSVC results show a very low misclassification rate, and big improvements are therefore not expected.

The *Iris* data have 4 variables and 3 classes. Here we label the first and second species as one class and compare this combined class to the third species. The *Checker* data are bivariate, so no selection of combinations of variables is necessary.

The *Bupa liver* data have 6 dimensions. We calculate all 15 combinations of pairwise models with $\mathbf{Z}_1$. These results vary greatly, and some combinations do not perform better than the additive model. As a second step we use models $\mathbf{Z}_2$ and $\mathbf{Z}_3$. The lowest misclassification rate is again lower than for the univariate case, and comparable to the *best* misclassification rate obtained with model $\mathbf{Z}_1$.

Table 3 displays the bivariate results. As in the univariate case the misclassification rate is the minimum of the mean classification rates, where the minimum is taken over the values of the smoothing parameter λ. The means are based on 10-fold cross validation, and the corresponding standard errors are given in brackets. The table also includes the 1-dimensional misclassifications obtained with PSVC. For the complete *Checker* example and for the *Bupa liver* data we have made use of the values given in Table 2 .

These preliminary results demonstrate that the misclassification rate can be reduced considerably when employing bivariate models.

5 Discussion

Support vector classifiers are increasingly used in classification problems. Pearce & Wand (2006) considered low-rank semiparametric regression models in the context of support vector classification such as kernels which arise from penalised splines. In this paper we examined computation issues relating to such penalised spline kernels, with emphasis on efficient interior point methods for support vector classification. Our comparison of different implementations showed that large savings in time and storage can be made when using custom built convex quadratic programming solvers for low-rank kernels.

For real data sets and univariate models we compared a number of different kernels and demonstrated that penalised spline kernels can perform as well as radial basis kernels. In addition, the penalised spline kernels enjoy an easy interpretability. Our preliminary results with bivariate models and penalised spline kernels show that further improvements in classification rates can be obtained when more complex models are considered.

Acknowledgement

Partial support has been provided by a University of New South Wales Faculty Research Grant.

References

Blake, C. and Merz, C. (1998). UCI repository of machine learning databases. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press.

Breiman, L. (2001). Statistical modeling: the two cultures (with discussion). *Statistical Science*, **16**, 199–231.

Burges, C. (1998). A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, **2**, 121–167.

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

Dudoit, S., Fridlyand, J. and Speed, T.P. (2002). Comparison of discrimination methods for the classification of tumors using gene expression data. *Journal of the American Statistical Association*, **97**, 77–87.

Eilers, P.H.C. and Marx, B.D. (1996). Flexible smoothing with B-splines and penalties (with discussion). *Statistical Science*, **11**, 89–121.

Ferris, M. C. and Munson, T. S. (2003). Interior point methods for massive support vector machines. *SIAM Journal on Optimization*, **13**, 783–804.

Fine, S. and Scheinberg, K. (2002). Efficient svm training using low-rank kernel representations. *Journal of Machine Learning Research*, **2**, 243–264.

French, J.L., Kammann, E.E. andWand, M.P. (2001). Comment on paper by Ke and Wang. *Journal of the American Statistical Association*, **96**, 1285–1288.

Hastie, T. (1996). Pseudosplines. *Journal of the Royal Statistical Society, Series B*, **58**, 379–396.

Hastie, T. J. and Tibshirani, R. J. (1990). *Generalized Additive Models*. Chapman & Hall/CRC

Hastie, T., Tibshirani, R. and Friedman, J. H. (2001). *The Elements of Statistical Learning, Data Mining, Inference, and Learning*. New York, NY: Springer.

Ho, T. K. and Kleinberg, E. M. (1996). *Building projectable classifiers of arbitrary complexity.* In Proceedings of the 13th International Conference on Pattern Recognition, 880–885, Vienna, Austria
`http://cm.bell-labs.com/who/tkh/pubs.html`.

Hush, D., Kelly, P., Scovel, C. and Steinwart, I. (2006). QP Algorithms with Guaranteed

Accuracy and Run Time for Support Vector Machines. *Journal of Machine Learning Research*, **7**, 733–769.

Johnson, M.E., Moore, L.M. and Ylvisaker, D. (1990). Minimax and maximin distance designs. *Journal of Statistical Planning and Inference*, **26**, 131–148.

Kaufman, L. and Rousseeuw, P.J. (1990). *Finding Groups in Data: An Introduction to Cluster Analysis*. New York: Wiley.

Mehrotra, S. (1992). On the Implementation of a Primal-Dual Interior Point Method. *SIAM Journal on Optimization*, **2**, 575–601.

Mészáros, Cs. (1998). The BPMPD interior point solver for convex quadratic programming problems. *Optimization Methods and Software*, **11&12**, 431–449.

Mészáros, Cs. (1999). Steplengths in infeasible primal-dual interior point methods of quadratic programming, *Operations Research Letters*, **25**, 39–45.

Nocedal, J. and Wright, S.J. (1999). *Numerical Optimization*. New York, NY: Springer.

Nychka, D., Haaland, P., O'Connell, M., Ellner, S. (1998). FUNFITS, data analysis and statistical tools for estimating functions. In *Case Studies in Environmental Statistics* (D. Nychka, W.W. Piegorsch, L.H. Cox, eds.), New York: Springer-Verlag, 159–179.

Nychka, D. & Saltzman, N. (1998). Design of Air Quality Monitoring Networks. In *Case Studies in Environmental Statistics* (D. Nychka, Cox, L., Piegorsch, W. eds.), Lecture Notes in Statistics, Springer-Verlag, 51–76.

Pearce, N.D. and Wand, M.P. (2006). Penalised Splines and Reproducing Kernel Methods. *The American Statistician*, **60**, 233–240.

Ruppert, D., Wand, M. P. and Carroll, R.J. (2003). *Semiparametric Regression*. New York: Cambridge University Press.

Schoenberg, I. (1968). Monosplines and quadrature formulae. In *Theory and Application of Spline Functions*, (Greville, T. ed.), Madison: University of Wisconsin Press.

Schölkopf, B. and Smola, A. J. (2002). *Learning with Kernels*. MIT Press, Cambridge, MA.

Simon, H. U. (2004). On the complexity of working set selection. In *Proceedings of the 15th International Conference on Algorithmic Learning Theory*. `http://eprints.pascal-network.org/archive/00000125/`.

Smola, A.J. and Schölkopf, B. (2000). Sparse greedy matrix approximation for machine learning. In *Proceedings of the 17th International Conference on Machine Learning*. San Francisco: Morgan Kaufmann.

Vandenberghe, L. and Comanor, K. (2003). A sequential analytic centering approach to the support vector machine. *In Proceedings of SPIE Advanced Signal Processing Algorithms, Architectures, and Implementations XIII*, 209–218, August 6-8, 2003, San Diego, California, USA.

Turlach, B.A. and Weingessel, A. (2006). `quadprog 1.4-8`. R package. `http://cran.r-project.org`.

Wahba, G. (1990). *Spline Models for Observational Data*. Philadelphia: SIAM.

Williams, C.K.I. and Seeger, M. (2001). Using the Nystrom method to speed up kernel machines. In *Advances in Neural Information Processing Systems*, **13**, (Leen, T.K. and Diettrich, T.G. eds.), 682–688, Cambridge USA: MIT Press.

Williams, C. and Seeger, M. (2001). Using the Nystroem Method to Speed Up Kernel Machines. *Neural Information Processing Systems*, **13**, 682–688.

Wood, S.N. (2003). Thin-plate regression splines. *Journal of the Royal Statistical Society, Series B*, **65**, 95–114.

Wright, S. J. (1997). *Primal-Dual Interior-Point Methods.* Philadelphia, PA: SIAM.

Yau, P., Kohn, R. and Wood, S. (2003). Bayesian variable selection and model averaging in high-dimensional multinomial nonparametric regression. *Journal of Computational and Graphical Statistics*, **12**, 1–32.

Appendix: R Code

This script demonstrates how to fit the spam dataset using a truncated linear spline kernel with the R package `LowRankQP`. Code for an interpretable plot (Figure 1) is also provided.

Load the data and normalise each variable.

```
library(LowRankQP); library(ElemStatLearn); data(spam)
n        <- nrow(spam)
use.vars <- c(5,6,7,8,16,17,25,26,27,37,45,46,52,53,56,57)
spam     <- spam[sample(1:n),c(use.vars,ncol(spam))]
d        <- ncol(spam) - 1
mu       <- mean(spam[,1:d])
sigma    <- sd(spam[,1:d],na.rm=TRUE)
x        <- (spam[,1:d] - mu)/sigma
y        <- 2*as.matrix((spam[,d+1]=="spam")+0) - 1
```

Create an R function to calculate the basis functions to be used.

```
CalculateBasis <- function(x,knots)
{
    X <- as.matrix(cbind(rep(1,nrow(x)),x))
    nKnots <- 0
    for (i in 1:ncol(x))
        nKnots <- nKnots + length(knots[[i]])
    Z  <- matrix(0,nrow(x),nKnots)
    s  <- 1
    for (i in 1:d)
    {
        Zi <- outer(x[,i],knots[[i]],"-")
        Z[,s:(s+length(knots[[i]])-1)] <- Zi*(Zi>0)
        s  <- s + length(knots[[i]])
    }
    list(X=X,Z=Z)
}
```

Set up the inputs for the quadratic program & solve using LowRankQP.

```
lambda <- 0.4279488; nKnots <- 20; knots  <- c()
for (i in 1:d)
    knots[[i]] <- quantile(unique(x[,i]),
                  seq(0,1,length=nKnots+2)[-c(1,nKnots+2)])
res1  <- CalculateBasis(x,knots)
A     <- res1$X*as.vector(y)
V     <- res1$Z*as.vector(y)/sqrt(2*lambda)
res2  <- LowRankQP(V,rep(-1,n),t(A),rep(0,ncol(A)),rep(1,n),
             method="SMW",verbose=TRUE,niter=200)
alpha <- t(V)%*%res2$alpha
beta  <- res2$beta
```

Set up labels and corresponding horizontal ranges for each variable.

```
data.labels <- c( "our","over","remove","internet","free",
    "business","hp","hpl","george","1999","re","edu","ch!",
    "ch$","CAPMAX","CAPTOT")
data.ran <- c()
data.ran[[1]]<-c(0,6);data.ran[[2]]<-c(0,4);data.ran[[3]]<-c(0,4);
data.ran[[4]]<-c(0,6);data.ran[[5]]<-c(0,10);data.ran[[6]]<-c(0,4);
data.ran[[7]]<-c(0,10);data.ran[[8]]<-c(0,10);data.ran[[9]] <-c(0,4);
data.ran[[10]]<-c(0,4);data.ran[[11]]<-c(0,10);data.ran[[12]]<-c(0,8);
data.ran[[13]]<-c(0,10);data.ran[[14]]<-c(0,4);
data.ran[[15]]<-c(0,1000);data.ran[[16]]<-c(0,4000)
```

Plot slices of the classification surface for different variables as described in Section 4.3.

```
n.plot      <- 300
median.data <- matrix(1,n.plot,1)%*%median(x)
par(mfrow=c(4,4))
for (i in 1:16)
{
    plot.x <- seq((data.ran[[i]][1]-mu[i])/sigma[i],
                  (data.ran[[i]][2]-mu[i])/sigma[i],length=n.plot)
    plot.data     <- median.data
    plot.data[,i] <- plot.x
    plot.x        <- sigma[i]*plot.x+mu[i]
    res3  <-CalculateBasis(plot.data,knots)
    plot.f<-res3$X%*%beta+(res3$Z/sqrt(2*lambda))%*%alpha
    plot(c(data.ran[[i]][1],data.ran[[i]][2]),
         c(-7.5,7.5),type="n",bty="l",xlab="",
         ylab="",ylim=c(-5,5),main=data.labels[i])
    lines(plot.x,plot.f,lwd=2,col="black")
    lines(plot.x,matrix(0,n.plot,1),lwd=0.5,col="black")
}
```