An Efficient Method for Gradient-Based Adaptation of Hyperparameters in SVM Models

S. Sathiya Keerthi
Yahoo! Research
Media Studios North
3333 Empire Avenue, Bldg. 4
Burbank, CA 91504
selvarak@yahoo-inc.com

Vikas Sindhwani
Department of Computer Science
University of Chicago
1100 E 58th Street
Chicago, IL 60637
vikass@cs.uchicago.edu

Olivier Chapelle
MPI for Biological Cybernetics
Dept. Schölkopf
Spemannstraße 38
72076 Tübingen
olivier.chapelle@tuebingen.mpg.de

Abstract

We consider the task of tuning hyperparameters in SVM models based on minimizing a smooth performance validation function, e.g., smoothed k-fold cross-validation error, using non-linear optimization techniques. The key computation in this approach is that of the gradient of the validation function with respect to hyperparameters. We show that for large-scale problems involving a wide choice of kernel-based models and validation functions, this computation can be very efficiently done; often within just a fraction of the training time. Empirical results show that a near-optimal set of hyperparameters can be identified by our approach with very few training rounds and gradient computations.

.

1 Introduction

Consider the general SVM classifier model in which, given n training examples $\{(x_i, y_i)\}_{i=1}^n$, the primal problem consists of solving the following problem:

$$\min_{(w,b)} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} l(o_i, y_i) \tag{1}$$

where l denotes a loss function over labels $y_i \in \{+1, -1\}$ and the outputs o_i on the training set. The machine's output o for any example x is given as:

$$o = w \cdot \phi(x) - b = \sum_{j=1}^{n} \alpha_j y_j k(x, x_i) - b \tag{2}$$

where b is the threshold parameter and, as usual, computations involving ϕ are handled using the kernel function: $k(x, z) = \phi(x) \cdot \phi(z)$. For example, the Gaussian kernel is given by

$$k(x, z) = \exp(-\gamma\|x - z\|^2) \tag{3}$$

The regularization parameter C and kernel parameters such as γ comprise the vector h of hyperparameters in the model. h is usually chosen by optimizing a validation measure (such as the k-fold cross validation error) on a grid of values (e.g. a uniform grid in the $(\log C, \log \gamma)$ space). Such a grid search is usually expensive. Particularly, when n is large, this search is so time-consuming that one usually resorts to either default hyperparameter values or crude search strategies. The problem becomes more acute when there are more than two hyperparameters. For example, for feature weighting/selection purposes one may wish to use the following ARD-Gaussian kernel:

$$k(x, z) = \exp(-\sum_t \gamma^t \|x^t - z^t\|^2) \tag{4}$$

where γ^t denotes the weight on the t^{th} feature denoted as x^t. In such cases, a grid based search of the hyperparameter space is ruled out.

In Figures 1, 2 (see section 6) we show contour plots of performance of an SVM on the $\log C - \log \gamma$ plane for a real-world binary classification problem. These plots show that learning performance behaves "nicely" as a function of hyperparameters. Intuitively, as C and γ are varied one expects the SVM to smoothly transition from providing underfitting solutions to overfitting solutions. Given that this phenomenon seems to occur routinely on real-world learning tasks[1], a very appealing and principled alternative to grid search is to consider a differentiable version of the performance validation function and invoke non-linear gradient-based optimization techniques for adapting hyperparameters. Such an approach requires the computation of the gradient of the validation function with respect to h.

Chapelle et al. (2002) give a number of possibilities for such an approach. One of their most promising methods is to use a differentiable version of the leave-one-out (LOO) error. A major expense in this method consists of the computation of the inverse of a kernel sub-matrix corresponding to the support vectors. (We will outline some details in section 3.) This is a bottleneck in large scale problems. Similar problems exist for gradient-based hyperparameter tuning procedures in Gaussian processes (Rasmussen and Williams (2006)).

We highlight the contributions of this paper.

1. We consider differentiable versions of validation-set-based objective functions for model selection (such as k-fold error) and give an efficient method for computing the gradient of this function with respect to h. Our method does not require the computation of the inverse of a large kernel submatrix. Instead, it only needs a single linear system of equations to be solved, which can be done either by decomposition or conjugate-gradient techniques. In essence, the cost of computing the gradient with respect to h is about the same, and usually much lesser than the cost of solving (1) for a given h.

2. Our method is applicable to a wide range of validation objective functions and SVM models that may involve many hyperparameters. For example, a variety of loss functions can be used together with multiclass classification, regression, structured output or semi-supervised SVM algorithms.

3. Large-scale empirical results show that with BFGS optimization, just trying about 10-20 hyperparameter points leads to the determination of optimal hyperparameters. Moreover, even as compared to a fine grid search, the gradient procedure provides a more precise placement of hyperparameters leading to better generalization performance. The benefit in terms of efficiency over the grid approach is evident even with just two hyperparameters. Particularly, in problems where the learning curve stabilizes only after several thousand examples, we expect our method to be of great value. In our empirical study we demonstrate the efficient tuning of C and γ_t's for large-scale binary classification problems; this is perhaps the most canonical hyperparameter tuning task in SVM classification. We also show the usefulness of our method for tuning more than two hyperparameters when optimizing other functions such as the F measure and weighted error rate. This is particularly useful for imbalanced problems.

This paper is organized as follows: In section 2, we discuss the general class of SVM models to which our method can be applied. In section 3, we discuss alternative approaches such as the LOO approach of (Chapelle et al. (2002)) and evidence maximization in Gaussian processes. In section 4, we describe our framework and provide the details of the gradient computation for general validation functions. In section 5, we discuss how to develop differentiable versions of several common

[1] For similar contour plots on other datasets, see Chapelle et al. (2002); Keerthi (2002).

performance validation functions. Empirical results are presented in section 6. We conclude this paper in section 7. The appendix outlines extensions of our method to other SVM models.

2 SVM Classification Models

In this section, we discuss the assumptions required for our method to be applicable. Consider SVM classification models of the form in (1).

Assumption 1. *The kernel function k is a continuously differentiable function of h.*

Consider three common SVM loss functions: (1) squared loss; (2) hinge loss; and (3) squared hinge loss. In each of these cases, the solution of (1) is obtained by computing the vector α that solves a dual problem. The solution usually leads to a linear system relating α and b:

$$P \begin{pmatrix} \alpha \\ b \end{pmatrix} = q \tag{5}$$

where P and q are, in general, functions of h.

We make the following assumption.

Assumption 2. *Locally around h (at which we are interested in calculating the gradient of the validation function to be defined soon) P and q are continuously differentiable functions of h.*

Let us now write down P and q for the three loss functions mentioned above, and discuss the validity of the above assumption.

Squared loss.

$$l(o_i, y_i) = (1 - y_i o_i)^2/2 \tag{6}$$

$$P = \begin{pmatrix} \lambda I + \Omega & -y \\ -y^T & 0 \end{pmatrix} \quad q = \begin{pmatrix} e \\ 0 \end{pmatrix} \tag{7}$$

where $\lambda = 1/C$, $\Omega_{ij} = y_i y_j K_{ij}$, I is the identity matrix, y is a vector containing the y_i, T denotes transpose, and e is a vector of all 1's. Assumption 2 always holds for squared loss.

Squared Hinge loss.

$$l(o_i, y_i) = \max\{0, 1 - y_i o_i\}^2/2 \tag{8}$$

After the solution of (1), the training set indices get partitioned into two sets: $I_0 = \{i : \alpha_i = 0\}$ and $I_u = \{i : \alpha_i > 0\}$. Then (5) is given by

$$\alpha_0 = 0, \quad P_u \begin{pmatrix} \alpha_u \\ b \end{pmatrix} = q_u \tag{9}$$

where α_0 is a vector containing $\{\alpha_i : i \in I_0\}$, α_u is a vector containing $\{\alpha_i : i \in I_u\}$,

$$P_u = \begin{pmatrix} \lambda I + \Omega_{uu} & -y_u \\ -y_u^T & 0 \end{pmatrix} \quad q = \begin{pmatrix} e_u \\ 0 \end{pmatrix} \tag{10}$$

and, Ω_{uu}, y_u and e_u are parallel to those for squared loss, but restricted to the indices in I_u. If the partitions I_0 and I_u do not change locally around a given h then assumption 2 holds. Generically, this happens for almost all h.

Hinge loss.

$$l(o_i, y_i) = \max\{0, 1 - y_i o_i\} \tag{11}$$

After the solution of (1), the training set indices get partitioned into three sets: $I_0 = \{i : \alpha_i = 0\}$, $I_c = \{i : \alpha_i = C\}$ and $I_u = \{i : 0 < \alpha_i < C\}$. Let α_0, α_c, α_u, y_c, y_u, e_c, e_u, Ω_{uc}, Ω_{uu} etc be appropriately defined vectors and matrices. Then (5) is given by

$$\alpha_0 = 0, \quad \alpha_c = C e_c, \quad \begin{pmatrix} \Omega_{uu} & -y_u \\ -y_u^T & 0 \end{pmatrix} \begin{pmatrix} \alpha_u \\ b \end{pmatrix} = \begin{pmatrix} e_u - \Omega_{uc}\alpha_c \\ y_c^T \alpha_c \end{pmatrix} \tag{12}$$

If the partitions I_0, I_c and I_u do not change locally around a given h then assumption 2 holds. Generically, this happens for almost all h.

The modified Huber loss function (Zhang, 2004) can also be used, though the derivation of (5) for it is more complex than for the three loss functions discussed above. Recently, weighted hinge loss with asymmetric margins (Grandvalet et al., 2005; Wu and Srihari, 2004) has been proposed for treating imbalanced problems and for incorporating prior knowledge.

Weighted Hinge loss.

$$l(o_i, y_i) = C_i \max\{0, m_i - y_i o_i\} \tag{13}$$

where $C_i = C_+$, $m_i = m_+$ if $y_i = 1$ and $C_i = C_-$, $m_i = m_-$ if $y_i = -1$.

Because C_+ and C_- are present, the hyperparameter C in (1) can be omitted. The SVM model with weighted hinge loss has four extra hyperparameters, C_+, C_-, m_+ and m_-, apart from the kernel hyperparameters. Our methods in this paper allow the possibility of efficiently tuning all these parameters together with kernel parameters.

The method described in this paper is not special to classification models only. It extends to a wide class of kernel methods for which the optimality conditions for minimizing a training objective function can be expressed (or well-approximated) as a linear system (5) in a continuously differentiable manner. We give a list of some of these models and discuss details for some of them in the appendix.

3 The Leave-One-Out (LOO) approach

Let us now briefly visit the LOO approach just to make a point regarding its computational effort. Take squared loss to begin with. Suppose we leave out the i-th example from the training set, train with the remaining examples and then use that solution to compute the *LOO validation output* v_i of the i-th example. It can be shown (see Opper and Winther (2000); Chapelle et al. (2002) for the SVM hard margin case) that v_i is given by

$$y_i v_i = 1 - \frac{\alpha_i}{(P^{-1})_{ii}} \tag{14}$$

where P is as in (7). The v_i thus obtained can be used to define smooth validation functions. In the case of squared hinge loss we need P_u^{-1} where P_u is as in (10). In the case of hinge loss LOO requires the inverse of the matrix

$$\begin{pmatrix} \Omega_{uu} & -y_u \\ -y_u^T & 0 \end{pmatrix} \tag{15}$$

Thus, the biggest disadvantage of the LOO based approach for large scale problems is that it requires the storage and inverse of a large matrix. For hinge and squared hinge losses, the size of this matrix is one more than the size of the non-bounded support vector set, I_u. Note that, even if, on some large scale problems, this set is of a manageable size at the optimal h, the corresponding set (which is dependent on h) can be large when h is away from the optimal; on many problems, such a far-off region is usually traversed during the adaptation process! To get an idea, consider the *Adult* dataset used in our empirical studies of section 6, where a gradient-based *k*-fold method (*Grad-Erate-1*) was used to determine the optimal hyperparameters C and γ of a SVM model using hinge loss and Gaussian kernel. Table 1 gives the number of non-bounded support vectors (*nSV*), i.e., the size of I_u, at various iterations of hyperparameter optimization. (Note that the size of the matrix in (15) is *nSV+1*.) Though *nSV* is not too big at the optimal hyperparameters (iteration 6), it is much bigger in the early iterations.

In Gaussian processes (Rasmussen and Williams (2006)) hyperparameters are tuned by maximizing the Evidence. The computation of the gradient of the Evidence with respect to the hyperparameters also requires the inverse of the kernel matrix associated with the training examples. Hence, like in LOO, hyperparameter optimization in large scale problems is expensive for Gaussian processes too.

Table 1: Average number of non-bounded support vectors (*nSV*) in 5-fold training at various iterations of *Grad-Erate-1* method on *Adult* with 16000 training examples.

Iteration	1	2	3	4	5	6
nSV	8647	4490	2747	2089	1428	1241

4 The gradient of a validation function

Suppose that for the purpose of hyperparameter tuning, we are given a validation scheme involving a small number of (training set, validation set) partitions, such as: (1) using a single validation set, (2) k-fold cross validation, or (3) averaging over k randomly chosen (training set, validation set) partitions. Our method applies to any of these three schemes. To keep notations simple, we explain the ideas only for scheme (1) and expand on the other schemes towards the end of this section. Note that throughout the optimization process, the training-validation splits are fixed.

Let $\{\tilde{x}_l, \tilde{y}_l\}_{l=1}^{\tilde{n}}$ denote the validation set. Let $\tilde{K}_{li} = k(\tilde{x}_l, x_i)$ involving a kernel calculation between an element of a validation set with an element of the training set. The output on the l^{th} validation example is:

$$\tilde{o}_l = \sum_i \alpha_i y_i \tilde{K}_{li} - b \tag{16}$$

which, for convenience, we will rewrite as

$$\tilde{o}_l = \psi_l^T \beta \tag{17}$$

where β is a vector containing α and b, and ψ_l is a vector containing $y_i \tilde{K}_{li}$, $i = 1, \ldots, n$ and -1 as the last element (corresponding to b).

Let us suppose that the model selection problem is formulated as a non-linear optimization problem:

$$h^\star = \operatorname*{argmin}_h f(\tilde{o}_1, \ldots, \tilde{o}_{\tilde{n}}) \tag{18}$$

where f is a differentiable validation function of the outputs $\tilde{o}_l$ which implicitly depend on h. In the next section, we will outline the construction of such functions for criteria like error rate, F1 measure etc.

We now discuss the computation of $\nabla_h f$. Let θ denote a generic parameter in h and let us represent partial derivative of some quantity, say v, with respect to θ as $\dot{v}$.

Before writing down expressions for $\dot{f}$, let us discuss how to get $\dot{\beta}$. Differentiating (5) with respect to θ gives

$$P\dot{\beta} + \dot{P}\beta = \dot{q} \quad \Rightarrow \quad \dot{\beta} = P^{-1}(\dot{q} - \dot{P}\beta) \tag{19}$$

Now let us write down $\dot{f}$.

$$\dot{f} = \sum_{l=1}^{\tilde{n}} \frac{\partial f}{\partial \tilde{o}_l} \dot{\tilde{o}}_l \tag{20}$$

where $\dot{\tilde{o}}_l$ is obtained by differentiating (17):

$$\dot{\tilde{o}}_l = \psi_l^T \dot{\beta} + \dot{\psi}_l^T \beta \tag{21}$$

The computation of $\dot{\beta}$ in (19) is the most expensive step, mainly because it requires P^{-1}. Note that, for hinge loss and squared hinge loss, P^{-1} can be computed in a somewhat cheaper way: only a matrix of the dimension of I_u needs to be inverted. Even then, in large scale problems the dimension of the matrix to be inverted can become so large that even storing it may be a problem; even when large storage is possible, the inverse can be very expensive. Most times, the effective rank of P is much smaller than its dimension. Thus, instead of computing $\dot{\beta} = P^{-1}(\dot{q} - \dot{P}\beta)$ in (19), we can instead solve

$$P\dot{\beta} = (\dot{q} - \dot{P}\beta) \tag{22}$$

for $\dot{\beta}$ approximately using decomposition methods or iterative methods such as conjugate-gradients. This can improve efficiency as well as take care of memory issues by storing P only partially and computing the remaining parts of P as and when needed.

Since the right-hand-side vector $(\dot{q} - \dot{P}\beta)$ in (22) changes for each different θ with respect to which we are differentiating, we need to solve (22) for each element of h. If the number of elements of h is not small (say, we want to use (4) with MNIST dataset which has more than 700 features) then, even with (22), the computations can still remain very expensive.

We now give a simple trick that shows that if the gradient calculations are re-organized, then *obtaining the solution of just a single linear system suffices for computing the full gradient of f with respect to all elements of h.*

Let us denote the coefficient of $\dot{\tilde{o}}_l$ in the expression for $\dot{f}$ in (20) by δ_l, i.e.,

$$\delta_l = \frac{\partial f}{\partial \tilde{o}_l} \tag{23}$$

Using (21) and plugging the expression for $\dot{\beta}$ from (19) into (20) gives

$$\begin{array}{rcl} \dot{f} & = & \sum_l \delta_l \dot{\tilde{o}}_l = \sum_l \delta_l \left(\psi_l^T P^{-1} (\dot{q} - \dot{P}\beta) + \dot{\psi}_l^T \beta \right) \\ & = & d^T (\dot{q} - \dot{P}\beta) + (\sum_l \delta_l \dot{\psi}_l)^T \beta \end{array} \tag{24}$$

where d is the solution of

$$P^T d = (\sum_l \delta_l \psi_l) \tag{25}$$

The beauty of the reorganization in (24) is that d *is the same* for all variables θ in h about which the differentiation is being done. Thus (25) needs to be solved only once. In concurrent work (Seeger, 2006) has used a similar idea for kernel logistic regression. In many ways, the gradient computation is much simpler and cleaner for the SVM models that we consider in this paper since the optimality conditions (5) are non-linear for kernel logistic regression.

As a word of caution, note that P may not be symmetric. See, e.g., the P arising from (12) for the hinge loss case. Also, when performing calculations, the parts corresponding to zero components should be omitted and the special structure of P should be utilized. To make these points clear, let us take the case of hinge loss. When computing $\dot{P}\beta$ the parts of $\dot{P}$ corresponding to α_0 can be ignored. Let (d_0, d_u, d_c, d_b) and (r_0, r_u, r_c, r_b) respectively denote the break-up of d and $(\sum_l \delta_l \psi_l)$. Then the structure in (12) can be utilized to get

$$\left(\begin{array}{cc} \Omega_{uu} & -y_u \\ -y_u^T & 0 \end{array} \right) \left(\begin{array}{c} d_u \\ d_b \end{array} \right) = \left(\begin{array}{c} r_u \\ r_b \end{array} \right) \quad d_c = r_c - \Omega_{uc}^T d_u + y_c d_b, \quad d_0 = 0 \tag{26}$$

The linear system in the above equation can be efficiently solved using conjugate gradient techniques similar to that in Chu et al. (2005).

The sequence of the computation of the full gradient of f with respect to h is as follows. First compute δ_l from (23). For various choices of validation function, we outline this computation in the next section. Then solve (25) for d. Then, for each θ use (24) to get all the derivatives of f. The computation of $\dot{P}\beta$ has to be performed for each hyperparameter separately. In problems with many hyperparameters, this is the most expensive part of the gradient computation. Note that in some cases, e.g., $\theta = C$, $\dot{P}\beta$ is immediately obtained. For $\theta = \gamma$ or γ_t, when using (3,4), one can cache pairwise distance computations while computing the kernel matrix. We have found that the cost of computing the gradient of f with respect to h to be usually much less than the cost of solving (1) and then obtaining f.

We can also employ the above ideas in a validation scheme where one uses k training-validation splits (e.g in k-fold cross-validation). In this case, for each partition one obtains the linear system (5), corresponding validation outputs (17) and the linear system in (25). The gradient is simply computed by summing over the k partitions, i.e., $\dot{f} = \sum_{j=1}^{k} \dot{f}^{(k)}$ where $\dot{f}^{(k)}$ is given by (24) using the quantities P, q, d etc associated with the k^{th} partition.

The model selection problem (18) may now be solved using, e.g., Quasi-Newton methods such as BFGS which only require function value and gradient at a hyperparameter setting. The suggestions made in Chung et al. (2003); Keerthi (2002) for effective application of Quasi-Newton methods to hyperparameter optimization can be used for our method too. In particular, reaching the minimizer of f too closely is not important. In our implementations we terminate optimization iterations when the following loose termination criterion is met:

$$|f(h^{k+1}) - f(h^k)| \leq 10^{-3} |f(h^k)| \tag{27}$$

where h^{k+1} and h^k are consecutive iterates in the hyperparameter optimization process.

Remark 1. For linear kernel, i.e., $k(x_i, x_j) = x_i \cdot x_j$, efficient direct primal methods for obtaining (w, b) without resorting to α and the dual problem (Keerthi and DeCoste, 2005) can be used. In such a case we can replace (5) by a primal linear system with (w, b) as the variables. All the computations and tricks of this section carry over easily. It is also possible to make good use of any special structures. For example, in the solution of large scale text classification by linear SVMs (Keerthi and DeCoste, 2005), both, the number of examples and the number of features can be large, but the data matrix is very sparse. For such a case, if (5) with (w, b) is used, then the linear system corresponding to (25) involves a large-but-sparse matrix, and, conjugate gradient techniques are very efficient for solving such a system. The ideas for linear kernel can be even extended to special methods involving nonlinear kernels, such as the Reduced SVM technique (Lee and Mangasarian, 2001; Lin and Lin, 2003) where a subset of the kernel basis functions is chosen and w is expressed as a linear combination of these basis functions.

Remark 2. A general concern with descent methods is the presence of local minima. In section 6, we make some encouraging empirical observations in this regard, e.g., local minima problems did not occur for the C, γ tuning task; for several other tasks, starting points that work surprisingly well could be easily obtained.

5 Smooth validation functions

We first recall some commonly used validation functions. Table 2 shows the confusion matrix parametrized by two entries: the number of true positives (tp) and the number of false positives (fp) for a binary classification problem with $\tilde{n}_+$ positive and $\tilde{n}_-$ negative examples. Consider validation functions that are general functions of the confusion matrix:

$$f = f(tp, fp) \tag{28}$$

Table 2: Confusion Matrix

pred $\rightarrow$ true $\downarrow$	+1	-1
+1	tp	$\tilde{n}_+ - tp$
-1	fp	$\tilde{n}_- - fp$

Let $u(z)$ denote the unit step function which is 0 when $z < 0$ and 1 otherwise. Denote $\tilde{u}_l = u(\tilde{y}_l \tilde{o}_l)$, which evaluates to 1 if the l^{th} example is correctly classified and 0 otherwise. Then, tp and fp can be written as,

$$tp = \sum_{l:\tilde{y}_l=+1} \tilde{u}_l, \qquad fp = \sum_{l:\tilde{y}_l=-1} (1 - \tilde{u}_l) \tag{29}$$

Error rate (er) is simply the percentage of incorrect predictions, i.e., $er = \frac{\tilde{n}_+ - tp + fp}{\tilde{n}}$.

For classification problems with imbalanced classes it is usual to define the validation function to be a weighted version of error rate or a function of *precision* and *recall*.

Weighted Error rate is defined in terms of differential costs for the two types of errors, i.e., $wer = \frac{(\tilde{n}_+ - tp) + \eta fp}{(\tilde{n}_+ + \eta \tilde{n}_-)}$, where η is the ratio of the cost of misclassifications of the negative class to that of the positive class.

Precision (pr) is defined as the percentage of positive predictions that are correct, i.e., $pr = \frac{tp}{tp+fp}$.

Recall (re) is defined as the percentage of positive validation examples that are correctly predicted, i.e., $re = \frac{tp}{\tilde{n}_+}$.

F measure (F) is defined as the harmonic mean of precision and recall, i.e., $F = \frac{2pr\ re}{pr+re} = \frac{2tp}{\tilde{n}_+ + tp + fp}$

In special problems one may want to maximize precision under a recall constraint:

$$\text{maximize } pr \quad \text{subject to: } re \geq re_{\min} \tag{30}$$

Area under ROC Curve (auc) can be computed as:

$$auc = \frac{1}{\tilde{n}_+ \tilde{n}_-} \sum_{j:\tilde{y}_j=1} \sum_{i:\tilde{y}_i=-1} u(\tilde{o}_j - \tilde{o}_i) \tag{31}$$

The subsampling scheme of Herschtal and Raskutti (2004) can be used to reduce the $O(\tilde{n}^2)$ calculation above to $O(\tilde{n})$ complexity.

It is common practice to evaluate measures like precision, recall and F measure while varying the threshold on the real-valued classifier output, i.e., at any given threshold σ_0, tp and fp can be redefined in terms of

$$\tilde{u}_l = u\left(\tilde{y}_l(\tilde{o}_l - \sigma_0)\right) \tag{32}$$

Note that as the threshold is varied from highly positive to highly negative values, (tp, fp) goes from $(0, 0)$ to (n_+, n_-). For imbalanced problems one may wish to maximize a score such as the F measure over all values of σ_0. In such cases, it is appropriate to incorporate σ_0 as an additional hyperparameter that needs to be tuned. Such bias-shifting is particularly also useful as a compensation mechanism for the mismatch between training objective function and validation function; often one uses an SVM as the underlying classifier even though it is not explicitly trained to minimize the validation function that the practioner truely cares about. In section 6, we make some empirical observations related to this point.

With σ_0 as a hyperparameter, it is also sensible to formulate an optimization problem to maximize **precision-recall breakeven** point:

$$\text{maximize } re \quad \text{subject to: } pr = re \tag{33}$$

The validation functions discussed above are based on discrete counts. In order to use gradient-based methods smooth functions of h are needed. We now develop smooth versions of validation optimization problems and for these give expressions for δ_l (23).

5.1 Sigmoidal Approximation

Let $\tilde{s}_l$ denote a sigmoidal approximation to $\tilde{u}_l$ (32) of the following form:

$$\tilde{s}_l = \frac{1}{1 + \exp\left(-\sigma_1 \tilde{y}_l \left(\tilde{o}_l - \sigma_0\right)\right)} \tag{34}$$

where $\sigma_1 > 0$ is a sigmoidal scale factor. *In general, σ_0, σ_1 may be functions of the validation outputs.* As discussed in the previous section, one may alternatively wish to treat σ_0 as an additional hyperparameter. The scale factor σ_1 influences how closely $\tilde{s}_l$ approximates the step function $\tilde{u}_l$ and hence controls the degree of smoothness in building the sigmoidal approximation. As the hyperparameter space is probed, the magnitude of the outputs can vary quite a bit. σ_1 takes the scale of the outputs into account. In section 5.2 we discuss various methods to set σ_0, σ_1.

Given a sigmoidal approximation, we consider a general validation function that is defined in terms of $\{\tilde{u}_l\}$. We build a differentiable version of such a function by simply replacing $\tilde{u}_l$ by $\tilde{s}_l$. Thus, we have,

$$f = f(\tilde{s}_1 \ldots \tilde{s}_{\tilde{n}}) \tag{35}$$

The value of δ_l (23) is given by:

$$\delta_l = \frac{\partial f}{\partial \tilde{s}_l}\frac{\partial \tilde{s}_l}{\partial \tilde{o}_l} + \left(\sum_r \frac{\partial f}{\partial \tilde{s}_r}\frac{\partial \tilde{s}_r}{\partial \sigma_0}\right)\frac{\partial \sigma_0}{\partial \tilde{o}_l} + \left(\sum_r \frac{\partial f}{\partial \tilde{s}_r}\frac{\partial \tilde{s}_r}{\partial \sigma_1}\right)\frac{\partial \sigma_1}{\partial \tilde{o}_l} \tag{36}$$

where the partial derivatives of $\tilde{s}_l$ with respect to $\tilde{o}_l, \sigma_0, \sigma_1$ are given by

$$\frac{\partial \tilde{s}_l}{\partial \tilde{o}_l} = \tilde{s}_l(1 - \tilde{s}_l)\sigma_1 \tilde{y}_l, \qquad \frac{\partial \tilde{s}_r}{\partial \sigma_0} = -\tilde{s}_r(1 - \tilde{s}_r)\sigma_1 \tilde{y}_l, \qquad \frac{\partial \tilde{s}_r}{\partial \sigma_1} = \tilde{s}_r(1 - \tilde{s}_r)\tilde{y}_r(\tilde{o}_r - \sigma_0) \tag{37}$$

Below we consider the computation of $\frac{\partial f}{\partial \tilde{s}_l}$ for the choices of validation functions mentioned earlier. In section 5.2 we discuss the computation of the remaining partial derivatives $\frac{\partial \sigma_0}{\partial \tilde{o}_l}$ and $\frac{\partial \sigma_1}{\partial \tilde{o}_l}$.

With the sigmoidal approximation $tp \approx \sum_{l:\tilde{y}_l=+1} \tilde{s}_l$ and $fp \approx \sum_{l:\tilde{y}_l=-1}(1-\tilde{s}_l)$, we can compute the following quantity for the validation function $f = f(tp, fp)$ (28).

$$\frac{\partial f}{\partial \tilde{s}_l} = \frac{\partial f}{\partial tp}\frac{\partial tp}{\partial \tilde{s}_l} + \frac{\partial f}{\partial fp}\frac{\partial fp}{\partial \tilde{s}_l} \tag{38}$$

To compute (38) we require the partial derivatives of f with respect to tp, fp. Let us write down the expressions for these terms for some common functions.

Error Rate $\frac{\partial er}{\partial tp} = -\frac{1}{\tilde{n}} \quad , \quad \frac{\partial er}{\partial fp} = \frac{1}{\tilde{n}}$ (39)

Weighted Error Rate $\frac{\partial wer}{\partial tp} = -\frac{1}{\tilde{n}_+ + \eta\tilde{n}_-} \quad , \quad \frac{\partial wer}{\partial fp} = \frac{\eta}{\tilde{n}_+ + \eta\tilde{n}_-}$ (40)

Precision $\frac{\partial pr}{\partial tp} = \frac{fp}{(tp+fp)^2} \quad , \quad \frac{\partial pr}{\partial fp} = -\frac{tp}{(tp+fp)^2}$ (41)

Recall $\frac{\partial re}{\partial tp} = \frac{1}{\tilde{n}_+} \quad , \quad \frac{\partial re}{\partial fp} = 0$ (42)

F measure $\frac{\partial F}{\partial tp} = \frac{2(\tilde{n}_+ + fp)}{(\tilde{n}_+ + tp + fp)^2} \quad , \quad \frac{\partial F}{\partial fp} = -\frac{2tp}{(\tilde{n}_+ + tp + fp)^2}$ (43)

The problem (30) can be approached either by separately writing the gradient expressions of pr and re with respect to h and feeding them to a constrained optimization method to solve (30), or, maximize $pr + \nu\, re$ for several values of ν and choose the one that achieves $re = re_{\min}$. The problem (33) can be similarly solved.

Area under ROC Curve (auc) Similar to what we did earlier, we can introduce new variables $\tilde{z}_{ji} = \tilde{o}_j - \tilde{o}_i$, relax the step function to a sigmoid over the $\tilde{z}_{ji}$ variables and derive expressions for $\delta_l = \frac{\partial auc}{\partial \tilde{o}_l}$. For auc the threshold parameter σ_0 is unnecessary.

5.2 Sigmoidal Smoothing Methods

We now discuss three methods to compute the sigmoidal parameters σ_0, σ_1 and calculate their partial derivatives with respect to $\tilde{o}$. In section 6, we will empirically compare these methods.

Direct Method

Here, we simply set,

$$\sigma_0 = 0, \qquad \sigma_1 = \frac{t}{\rho} \tag{44}$$

where ρ denotes standard deviation of the outputs $\{\tilde{o}_l\}$, i.e., $\rho = \sqrt{\frac{\sum_k (\tilde{o}_k - \mu)^2}{n}}$, $\mu = \frac{1}{\tilde{n}}\sum_k \tilde{o}_k$; and t is a constant which is heuristically set to some fixed value in order to well-approximate the step function. In our implementation we use $t = 10$. The partial derivatives required to compute (36) are given by,

$$\frac{\partial \sigma_0}{\partial \tilde{o}_l} = 0 \qquad \frac{\partial \sigma_1}{\partial \tilde{o}_l} = -\frac{t}{n\rho^3}(\tilde{o}_l - \mu) \tag{45}$$

Hyperparameter Bias Method

Here, we consider σ_0 as a hyperparameter and set σ_1 as above. Note that in this case the $\tilde{o}$ do not depend on σ_0 and one can write the gradient component with respect to σ_0 simply as,

$$\frac{\partial f}{\partial \sigma_0} = \sum_r \frac{\partial f}{\partial \tilde{s}_r}\frac{\partial \tilde{s}_r}{\partial \sigma_0} \tag{46}$$

instead of using (20). For other gradient components, we simply use (45) together with (36) and (20).

Minimization Method

In this method, we obtain σ_0, σ_1 by performing sigmoidal fitting based on unconstrained minimization of some smooth criterion N, i.e.,

$$(\sigma_0, \sigma_1) = \underset{\Re^2}{\operatorname{argmin}}\, N \tag{47}$$

Denote the gradient (g) and Hessian (H) of N in with respect to $(\sigma_1, \sigma_0)^T$ as:

$$g = \begin{pmatrix} g_1 \\ \\ g_2 \end{pmatrix}, \quad H = \begin{pmatrix} H_1 & H_0 \\ H_0 & H_2 \end{pmatrix} \tag{48}$$

The optimality conditions for minimizing N with respect to $(\sigma_1, \sigma_0)^T$ is nothing but $g = 0$. We can take the derivative of this system with respect to $\tilde{o}_l$ to get

$$H \begin{pmatrix} \frac{\partial \sigma_1}{\partial \tilde{o}_l} \\ \frac{\partial \sigma_0}{\partial \tilde{o}_l} \end{pmatrix} + \begin{pmatrix} \frac{\partial g_1}{\partial \tilde{o}_l} \\ \frac{\partial g_2}{\partial \tilde{o}_l} \end{pmatrix} = 0 \tag{49}$$

The small 2×2 linear system in (49) can be analytically solved. In particular, we obtain the following:

$$\frac{\partial \sigma_1}{\partial \tilde{o}_l} = \frac{1}{\Delta}\left(H_2 \frac{\partial g_1}{\partial \tilde{o}_l} - H_0 \frac{\partial g_2}{\partial \tilde{o}_l}\right), \quad \frac{\partial \sigma_0}{\partial \tilde{o}_l} = \frac{1}{\Delta}\left(-H_0 \frac{\partial g_1}{\partial \tilde{o}_l} + H_1 \frac{\partial g_2}{\partial \tilde{o}_l}\right) \tag{50}$$

where $\Delta = H_0^2 - H_1 H_2$.

These expressions allow (36) to be computed for the this method. Note that when f and N coincide, $g = 0$ implies that the last two terms in (36) evaluate to zero.

A natural choice of N is based on Platt's method (Platt (1999); Lin et al. (2003)) where $\tilde{s}_l$ is intepreted as the posterior probability that the class of l^{th} validation example if $\tilde{y}_l$. Then the sigmoid parameters σ_1 and σ_0 may be obtained by minimizing the **negative log-likelihood** function

$$N = -\sum_l \log(\tilde{s}_l) \tag{51}$$

In practice, as derived from Bayesian considerations in Platt (1999); Lin et al. (2003), one makes a small allowance for the opposite label to be true,

$$N = -\sum_l \tilde{t}_l \log(\tilde{s}_l) + (1 - \tilde{t}_l)\log(1 - \tilde{s}_l) \tag{52}$$

$$\tilde{t}_l = \begin{cases} (\tilde{n}_+ + 1)/(\tilde{n}_+ + 2) & \text{if } \tilde{y}_l = +1 \\ (\tilde{n}_- + 1)/(\tilde{n}_- + 2) & \text{if } \tilde{y}_l = -1 \end{cases}$$

for the $\tilde{n}_+$ positive and $\tilde{n}_-$ negative examples respectively. This amounts to adding regularization to avoid overfitting σ_1 and σ_0. N can be minimized with respect to $(\sigma_1, \sigma_0)^T$ using Newton's method with backtracking line search following the pseudo-code given in Lin et al. (2003). Sigmoidal fitting based on the negative log-likelihood function has also been previously proposed in Chapelle et al. (2002). Below, we record some expressions we need to compute (36) for this choice.

$$g = \begin{pmatrix} \sum_l \tilde{y}_l \tilde{o}_l (\tilde{s}_l - \tilde{t}_l) \\ \\ \sum_l (\tilde{t}_l - \tilde{s}_l)\sigma_1 \tilde{y}_l \end{pmatrix}, \quad H = \begin{pmatrix} \sum_l \tilde{o}_l^2 \tilde{s}_l (1 - \tilde{s}_l) & -\sum_l \sigma_1 \tilde{o}_l \tilde{s}_l (1 - \tilde{s}_l) \\ -\sum_l \sigma_1 \tilde{o}_l \tilde{s}_l (1 - \tilde{s}_l) & \sum_l \sigma_1^2 \tilde{s}_l (1 - \tilde{s}_l) \end{pmatrix} \tag{53}$$

$$\frac{\partial g_1}{\partial \tilde{o}_l} = \tilde{y}_l(\tilde{s}_l - \tilde{t}_l) + \sigma_1 \tilde{y}_l \tilde{o}_l \tilde{s}_l (1 - \tilde{s}_l), \quad \frac{\partial g_2}{\partial \tilde{o}_l} = -\sigma_1^2 \tilde{s}_l (1 - \tilde{s}_l) \tag{54}$$

In case $f = N$ as defined in (51), in (36) we can use,

$$\frac{\partial f}{\partial \tilde{s}_l} = \frac{(\tilde{s}_l - \tilde{t}_l)}{\tilde{s}_l (1 - \tilde{s}_l)} \tag{55}$$

Remark 3. If we are mainly interested in negative log-likelihood (51) as the measure of generalization performance, it is more appropriate to train a kernel logistic regression model instead of the SVM model. However, this complicates the gradient computations since the optimality conditions (5) for kernel logistic regression are non-linear. See Seeger (2006) for more details in this direction.

We can also use the σ_0 and σ_1 obtained using (47) and (52) to define the *probabilitstic error rate* (per):

$$per = \frac{1}{\tilde{n}} \sum_l (1 - \tilde{s}_l) \tag{56}$$

For the *Minimization method* another choice for N is the squared loss $N = \sum_l (\tilde{u}_l - \tilde{s}_l)^2$ for which computations similar to those for negative log-likelihood can be done.

6 Empirical Results

We demonstrate the effectiveness of our method on binary classification, mainly taking error rate as the validation/test objective function of interest. The SVM model with hinge loss was used. SVM training was done using the SMO algorithm. Five fold cross validation was used to form the validation objective functions. Four datasets were used in our study: *Adult*, *IJCNN*, *Vehicle* and *Splice*. The first three were taken from *http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/* and *Splice* was taken from *http://ida.first.fraunhofer.de/~raetsch/*. The total number of examples and the number of features in these datasets is given in Table 3. For each dataset training sets of different sizes were chosen in a class-wise stratified fashion; the remaining examples formed the test set.

Table 3: Properties of various datasets.

	Adult	*IJCNN*	*Vehicle*	*Splice*
Total examples	32561	141691	98528	3175
Num of features	123	22	100	60

The Gaussian kernel (3) and the ARD-Gaussian kernel (4) were used. In the case of the Gaussian kernel, for which the SVM model has the two hyperparameters C and γ, we also tried the popular *Grid* approach of searching over a grid of values in the (C, γ) space to minimize validation error rate. For each of these two parameters we tried 15 values: 2^{imin+i}, $i = 0, \ldots, 14$. The value of $imin$ was chosen differently for C and γ; for each of the datasets, these $imin$ values were chosen so that the optimal hyperparameter values were in the middle of the region covered by the grid. To be efficient, the solution (α) from one hyperparameter vector was seeded as the starting point of SMO for the nearby hyperparameter vector.

For our gradient based methods, we used the starting point, $C = 1$ and $\gamma = 1$ for the hyperparameter optimization process when the Gaussian kernel was used. In the case of the ARD-Gaussian kernel, we did the following. We first optimized the validation function using the Gaussian kernel, obtained the optimal $(C^\star, \gamma^\star)$ and then continued the optimization with the ARD-Gaussian kernel, using $C = C^\star$ and $\gamma^t = \gamma^\star \; \forall t$ as the starting point.

Comparison of validation functions. For minimizing error rate, we can use er of section 5 together with the *Direct method* of section 5.2. Let us refer to this approach as *Grad-Erate-1*. An alternative is to use the probabilistic estimate of error rate in (56). Let us refer to this approach as *Grad-Erate-2*. We compare the performance of these validation functions using the *IJCNN* dataset. For a training set size of 2000, Figure 1 gives the contours of: (1) actual validation error rate based on counting misclassifications; (2) er, the smooth validation error rate based on the *Direct method*; (3) per, the probabilistic error rate of (56); and (4) actual test error rate. The plots also show the sequence of points obtained by the optimization process by *Grad-Erate-1* and *Grad-Erate-2* as well as the best point produced by the *Grid* method. Clearly, *Grad-Erate-1* and *Grid* perform better than *Grad-Erate-2*. A comparison of the contours shows that er is a better representation of the generalization error than per; also er is a very nicely smoothed representation of the (discontinuous) validation error rate. Figure 2 shows the corresponding contour plots for a much larger training set of size 8000. With such a large training set, the contours of test error rate, those of validation error rate and those of er become closer, while the contours of per are still somewhat shifted away. We also studied

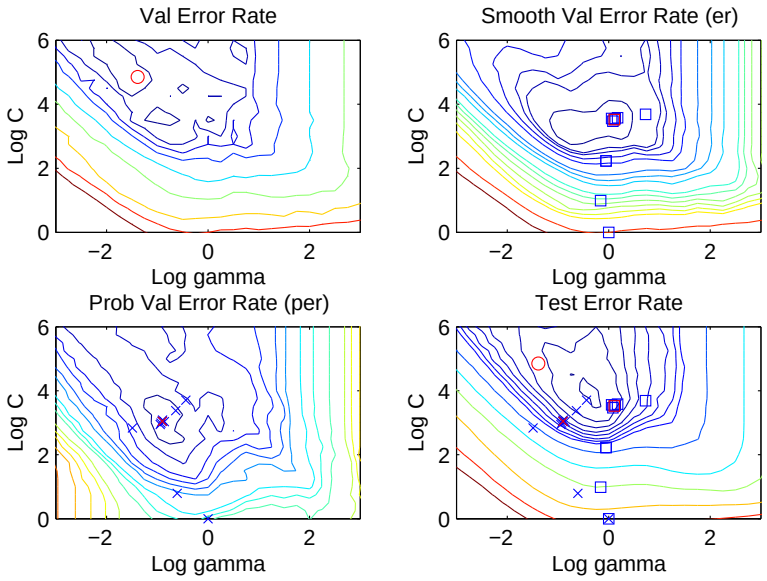

Figure 1: Various contours for *IJCNN* with 2000 training points. The sequence of points generated by *Grad-Erate-1* and *Grad-Erate-2* are shown, respectively by the symbols, □ and ×. The best point produced by *Grid* is shown by ○. All optimal points are marked in red.

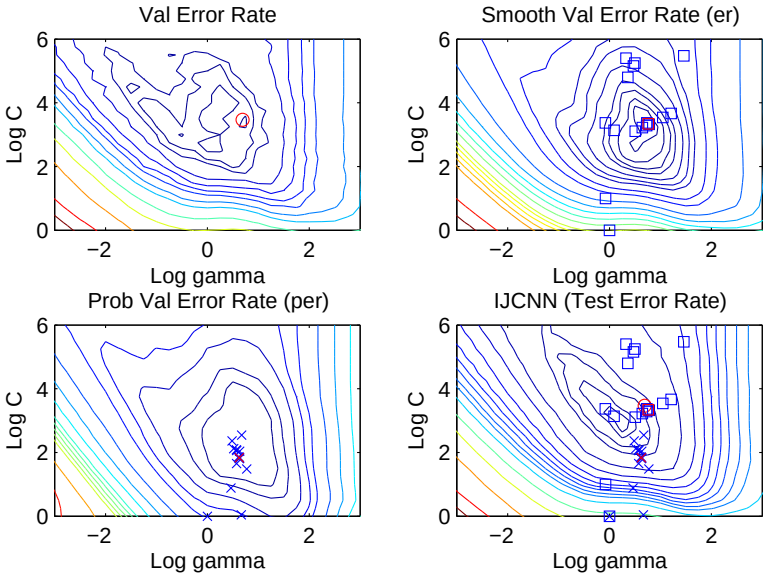

Figure 2: Various contours for IJCNN with 8000 training points. The sequence of points generated by *Grad-Erate-1* and *Grad-Erate-2* are shown, respectively by the symbols, □ and ×. The best point produced by *Grid* is shown by ○. All optimal points are marked in red.

the contours of negative log-likelihood (51); this function tends to behave very similar to per. The sequence of points generated by the gradient-based method using (51) (call it as the *Grad-LogLik* method) is also very close to that of *Grad-Erate-2*.

Comparison of Grid and Grad methods. For various training set sizes of the *IJCNN* dataset we compare the speed and generalization performance of *Grid*, *Grad-Erate-1*, *Grad-Erate-2* and *Grad-*

LogLik. Table 4 gives the results. Clearly the gradient-based methods are much more efficient than *Grid*. The good speed improvement is seen even at small training set sizes. Although the efficiency of *Grid* can be improved in certain ways (say, by doing a crude search followed by a refined search, by avoiding unnecessary exploration of difficult regions in the hyperparameter space etc) the gradient-based methods are still much better because of their ability to systematically plunge in hyperparameter space to precisely fix the hyperparameters at the optimal locations. The test error rates of Table 4 also quantify our earlier finding that, for minimizing error rate *Grad-Erate-1* is a better method than *Grad-Erate-2* and *Grad-LogLik*. Table 5 compares *Grid* and *Grad-Erate-1* on *Adult* and *Vehicle* datasets for various training set sizes. Though the generalization performance of the two methods are close, *Grid* is much slower.

Table 4: Comparison of *Grid*, *Grad-Erate-1*, *Grad-Erate-2* and *Grad-LogLik* on *IJCNN*. *nf* is the number of hyperparameter vectors tried. (For *Grid*, *nf* is 225.) *cpu* is computational time in minutes. *erate* is the percentage test error rate.

	Grid		*Grad-Erate-1*			*Grad-Erate-2*			*Grad-LogLik*		
$n_{\rm trg}$	*cpu*	*erate*	*nf*	*cpu*	*erate*	*nf*	*cpu*	*erate*	*nf*	*cpu*	*erate*
2000	10.03	2.95	11	4.58	2.87	11	3.21	3.12	12	3.97	3.16
4000	38.77	2.42	12	11.40	2.42	12	12.04	2.74	11	11.62	2.75
8000	218.92	1.76	14	68.58	1.77	15	53.25	1.94	14	50.72	1.94
16000	1130.37	1.24	12	127.03	1.26	13	137.07	1.30	12	130.08	1.32
32000	5331.15	0.91	9	382.20	0.91	11	410.76	0.96	12	437.98	0.97

Table 5: Comparison of *Grad-Erate-1* and *Grid* methods on *Adult* and *Vehicle*. *nf* is the number of hyperparameter vectors tried. (For *Grid*, *nf* is 225.) *cpu* is computational time in minutes. *erate* is the percentage test error rate. For *Vehicle* and $n_{\rm trg}$=16000, the solution of *Grid* was discontinued after 5 days of computation.

	Adult					*Vehicle*				
	Grad-Erate-1			*Grid*		*Grad-Erate-1*			*Grid*	
$n_{\rm trg}$	*nf*	*cpu*	*erate*	*cpu*	*erate*	*nf*	*cpu*	*erate*	*cpu*	*erate*
2000	9	3.62	16.21	8.66	16.14	7	2.50	13.58	15.25	13.84
4000	16	15.98	15.64	37.53	15.95	5	8.60	13.29	135.28	13.30
8000	10	52.17	15.69	306.25	15.59	9	83.10	12.84	1458.12	12.82
16000	6	256.40	15.40	3667.90	15.37	6	360.88	12.58	–	–

Feature Weighting Experiments. To study the effectiveness of our gradient-based approach when many hyperparameters are present, we use the ARD-Gaussian kernel in (4) and tune C together with all the γ^t's. Like in *Grad-Erate-1*, the smooth estimate er based on *Direct method* is used. As mentioned earlier, the solution of *Grad-Erate-1* was used to initialize the solution. We denote the resulting method as *Grad-ARD*. Table 6 gives the results for various training set sizes of *IJCNN*. *Grad-ARD* achieves a decent improvement in generalization performance over *Grad-Erate-1* without increasing the computational cost by much. Table 7 gives the results of *Splice* for the training set size of 2000. In spite of the fact that the number of hyperparameters tuned is large (i.e., 61), the extra cpu time is not excessive. What is remarkable is the huge improvement in generalization performance that is achieved by allowing individual feature weights to be tuned. In large scale problems where such significant gains are possible, our gradient-based methods with the ARD-Gaussian kernel are very valuable.

Improving F-measure by threshold adjustment. In section 5 we mentioned about the possible value of threshold adjustment when the validation/test function of interest is a quantity that is different from error rate. We now illustrate this by taking the *Adult* dataset, with *F-measure* as the quantity of interest. The size of the training set is 2000. Gaussian kernel (3) was used. For smoothening we employed the direct method of section 5.2. We implemented two methods: in the first method we set $\sigma_0 = 0$ and tuned only C and γ; in the second method we tuned the three hyperparameters C, γ and σ_0. For the first method we started the optimization from $C = 1$, $\gamma = 1$ as usual. After obtaining its optimizer $(C^\star, \gamma^\star)$, we started the second method with $C = C^\star$, $\gamma = \gamma^\star$ and $\sigma_0 = 0$. We ran the methods on ten different randomly chosen training set/test set splits. Table 8 gives the statistics of

Table 6: Comparison of *Grad-Erate-1*, *Grid* and *Grad-ARD* (*Grad-Erate-1* with feature weighting) on *IJCNN*. *nf* is the number of hyperparameter vectors tried. (For *Grid*, *nf* is 225.) *cpu* is computational time in minutes. (The solution of *Grad-ARD* was seeded using the solution of *Grad*. The *cpu* given is the extra time needed for doing this optimization.) *erate* is the percentage test error rate.

	Grad-Erate-1			*Grid*		*Grad-ARD*		
$n_{\rm trg}$	*nf*	*cpu*	*erate*	*cpu*	*erate*	*nf*	*cpu*	*erate*
2000	11	4.58	2.87	10.03	2.95	28	5.63	2.65
4000	12	11.40	2.42	38.77	2.42	13	8.40	2.14
8000	14	68.58	1.77	218.92	1.76	17	38.58	1.50
16000	12	127.03	1.26	1130.37	1.24	20	154.03	1.08
32000	9	382.20	0.91	5331.15	0.91	7	269.16	0.82

Table 7: Comparison of *Grad-Erate-1*, *Grid* and *Grad-ARD* (*Grad-Erate-1* with feature weighting) on *Splice*. *nf* is the number of hyperparameter vectors tried. (For *Grid*, *nf* is 225.) *cpu* is computational time in minutes. (The solution of *Grad-ARD* was seeded using the solution of *Grad*. The *cpu* given is the extra time needed for doing this optimization.) *erate* is the percentage test error rate.

	Grad-Erate-1			*Grid*		*Grad-ARD*		
$n_{\rm trg}$	*nf*	*cpu*	*erate*	*cpu*	*erate*	*nf*	*cpu*	*erate*
2000	13	7.57	8.17	11.42	9.19	37	35.04	3.49

F-measure values on 5-fold cross validation and on the test set. Clearly, the use of σ_0 yields a very significant improvement on the F-measure. To give a better idea we plot, in Figure 3, the variation of error rate and F-measure on validation and test, for one run. The SVM solution corresponds to zero threshold. Clearly, error rate has its minimum very close to zero threshold. On the other hand, F-measure has its maximum at a threshold value that is well shifted away from zero. Also, the F-measure value of the SVM solution (zero threshold) is significantly smaller than the maximum value of F-measure. In general, error rate and F-measure achieve their best values at different values of hyperparameters.

Table 8: Mean (standard deviation) of F-measure values on the *Adult* dataset.

	Without σ_0		*With* σ_0	
	Validation	Test	Validation	Test
F-measure	0.6385 (0.0062)	0.6363 (0.0081)	0.6635 (0.0095)	0.6641 (0.0044)

Optimizing weighted error rate in imbalanced problems. In imbalanced problems where the proportion of examples in the positive class is small, it is usual to minimize weighted error rate wer (see section 5) with a small value of η. One can think of four possible methods in which, apart from the Gaussian kernel parameter γ and threshold[2] σ_0, we include other parameters by considering sub-cases of the weighted hinge loss model of section 2 – (1) *Usual SVM:* Set $m_+ = m_- = 1$, $C_+ = C$, $C_- = C$ and tune C. (2) Set $m_+ = m_- = 1$, $C_+ = C$, $C_- = \eta C$ and tune C. (3) Set $m_+ = m_- = 1$ and tune C_+ and C_- treating them as independent parameters. (4) Use the full *Weighted Hinge loss* model of section 2 and tune C_+, C_-, m_+ and m_-. For the first two methods we started the optimization from $C = 1$, $\gamma = 1$. For the third method we started from $C_+ = C_- = 1$, $\gamma = 1$. After obtaining the optimizer $(C^\star, \gamma^\star)$ of method 1, we seeded the fourth method with $C_+ = C_- = C^\star$, $\gamma = \gamma^\star$ and $m_+ = m_- = 1$. For all four methods, the initial value of σ_0 was set to 0. To compare the performance of these methods we take the *IJCNN* dataset, randomly choosing 2000 training examples and keeping the remaining examples as the test set. Ten such random splits were tried. We take $\eta = 0.01$. The top half of Table 9 gives the statistics of weighted error rate values associated with validation and test. The weighted hinge loss model gives the best performance.

The presence of the threshold parameter σ_0 is important for the first three methods. Interestingly, for the weighted hinge loss method, tuning of threshold has little effect. Grandvalet et al. (2005) also

[2]We used the *Hyperparameter bias method* of section 5.2 for smoothening.

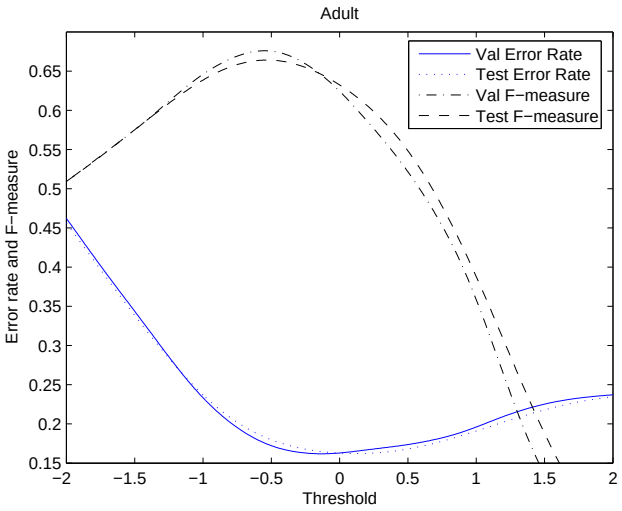

Figure 3: Plot of Error rate and F-measure versus threshold for the *Adult* dataset with 2000 training points and 30561 test points.

Table 9: Mean (standard deviation) of weighted (weight is 1 for positive class and $\eta = 0.01$ for negative class) error rate values on the *IJCNN* dataset.

	$C_+ = C, C_- = C$	$C_+ = C, C_- = \eta C$	C_+, C_- tuned	Full Weighted Hinge
	With σ_0			
Validation	0.0571 (0.0183)	0.0419 (0.0060)	0.0490 (0.0104)	0.0357 (0.0063)
Test	0.0638 (0.0160)	0.0549 (0.0098)	0.0571 (0.0136)	0.0461 (0.0078)
	Without σ_0			
Validation	0.1953 (0.0557)	0.1051 (0.0164)	0.1008 (0.0607)	0.0364 (0.0061)
Test	0.1861 (0.0540)	0.0897 (0.0154)	0.0969 (0.0502)	0.0469 (0.0076)

make the observation that this method places the threshold at the right spot on its own. The bottom half of Table 9 gives the performance statistics of the methods when threshold is not tuned. We also found that the weighted hinge loss method did not lose its good performance even when one of the margin parameters, m_+ is simply set to 1 and only the remaining parameters (C_+, C_-, m_- and γ) were tuned. The weighted margin hinge loss model seems to be very good for solving imbalanced problems.

Cost Break-up. In the gradient-based solution process, each step of the optimization requires the evaluation of f and $\nabla_h f$. In doing this, there are three steps that take up the bulk of the computational cost: (1) the solution of the training problem (determination of α) using the SMO algorithm; (2) the solution of the linear system in (25); and (3) the remaining computations associated with the gradient, of which the computation of $\dot{P}\beta$ in (24) is the major part. It is useful to have an idea of the break-up of these individual costs. Table 10 gives the relative break-up of the costs for the *IJCNN* dataset, for solution by *Grad-Erate-1* and *Grad-ARD* methods. Clearly, the cost of solution by SMO forms the major bulk of the total computational time. It is also encouraging to note that the $\dot{P}\beta$ cost of *Grad-ARD* doesn't become large in spite of the fact that 23 hyperparameters are tuned there. This is mainly due to the efficient usage of terms in the ARD-Gaussian calculations that we mentioned in section 4.

7 Conclusion

The main contribution of this paper is a fast method of computing the gradient of a validation function with respect to hyperparameters for a range of SVM models; together with a nonlinear opti-

Table 10: Cost break-up for *Grad-Erate-1* and *Grad-ARD* on *IJCNN*. We take the cost of computing the full gradient vector $\nabla_h f$ to be unity and give all other costs relative to that. *SMO* is the cost of SMO. *Lin* is the cost of solving the linear system in (25); $\dot{P}\beta$ is the cost of all remaining gradient computations, of which $\dot{P}\beta$ in (24) is the major part.

	Grad-Erate-1			*Grad-ARD*		
$n_{\rm trg}$	*SMO*	*Lin*	$\dot{P}\beta$	*SMO*	*Lin*	$\dot{P}\beta$
2000	6.5	0.26	0.74	6.6	0.32	0.68
4000	15.5	0.31	0.69	4.8	0.34	0.66
8000	10.2	0.29	0.71	4.9	0.38	0.62
16000	8.5	0.34	0.66	4.6	0.37	0.63
32000	9.4	0.31	0.69	5.2	0.34	0.66

mization technique such as BFGS it can be used to efficiently determine the optimal values of many hyperparameters. Even in models with just two hyperparameters our approach is faster and offers a more precise hyperparameter placement than the *Grid* approach, even in medium sized problems. Our approach is particularly of great value for large scale problems.

The ability to tune many hyperparameters easily should be used with care. On a text classification problem involving many thousands of features (see Remark 1 at the end of section 4) we placed an independent feature weight for each feature and optimized all these weights (together with C) only to find severe overfitting taking place. Therefore, for a given problem it is important to choose the set of hyperparameters carefully, in accordance with the amount of training examples available.

Appendix. Other SVM Models

In section 2 we discussed the SVM classification model and several associated loss functions. The expression of local optimality via (5) is the key property that allows the application of the ideas of section 3 to derive efficient expressions for the gradient of the validation objective function with respect to hyperparameters. We now show that (5) also holds for several other SVM models.

The derivation of (5) for regression models (squared loss, ϵ-insensitive loss etc) is quite easy. For the SVM ordinal regression models in Herbrich et al. (2000); Shashua and Levin (2000); Chu and Keerthi (2005) the derivation of (5) is very much along the lines of the SVM classification model.

Multi-class SVM models. There are several versions of these models. For multi-class models that involve a single optimization problem (Weston and Watkins, 1999; Crammer and Singer, 2005) the derivation of (5) is easy as it follows directly from the optimality conditions of the dual problem. But these models are rarely used in practice because of their expensive solution. More popular are models that use combinations of several binary models, such as *One-Vs-Rest*, *One-Vs-One* and *Error Correcting Codes*. We describe the details of the approach only for *One-Vs-Rest*. Similar ideas can be used for the other models.

Suppose there are m classes. In the *One-Vs-Rest* method, for each $c = 1, \ldots, m$ there is one binary model for differentiating class c from the rest, with the 'local' representation:

$$\tilde{o}_l^c = \psi_l^{cT} \beta^c, \quad P^c \beta^c = q^c \tag{57}$$

Here $\tilde{o}_l^c \ \forall \ l$ is the set of validation outputs. Let us say this binary model is set up in such a way that $\tilde{o}_l^c > 0$ denotes class c and $\tilde{o}_l^c < 0$ denotes the remaining classes. In the presence of several independent hyperparameters (e.g., a different C value for each binary model) it is appropriate to define normalized outputs,

$$\hat{o}_l^c = \tilde{o}_l^c / \sigma^c \tag{58}$$

where σ^c is the standard deviation of $\{\tilde{o}_l^c\}_{l=1}^m$ (like it was done in (34) and (44)). Then the multi-class decision function for the l-th validation example can be taken as

$$\text{class } = \arg\max_c \hat{o}_l^c \tag{59}$$

The following soft-max function can be used to approximate this function smoothly:

$$\text{class } = \arg\max_c \hat{s}_l^c, \quad \text{where} \quad \hat{s}_l^c = \frac{\exp\left(t\hat{o}_l^c\right)}{\sum_y \exp\left(t\hat{o}_l^y\right)} \tag{60}$$

where t is a constant, say $t = 10$. Suppose, for example l, the correct class is c^l. The validation error function can be approximated as

$$f = \sum_l (1 - \hat{s}_l^{c^l}) \tag{61}$$

Parallel to (20) we can write

$$\dot{f} = -\sum_l \sum_c \frac{\partial \hat{s}_l^{c^l}}{\partial \hat{o}_l^c} \left[\frac{\partial \hat{o}_l^c}{\partial \tilde{o}_l^c} + \frac{\partial \hat{o}_l^c}{\partial \sigma^c} \frac{\partial \sigma^c}{\partial \tilde{o}_l^c} \right] \dot{\tilde{o}}_l \tag{62}$$

The remaining calculations go as in section 3. Thus, the only added complication in the gradient expressions is that, everywhere there is a summation term over all m models.

The above ideas for gradient computation can be easily extended to other multi-class methods that combine binary models, and even to other situations (such as those in (Collobert et al., 2002)) where several SVM models are combined to form a single overall model.

Multivariate Losses and Structured Output models. The gradient calculations of section 3 can also be used for structured output models such as those in Tsochantaridis et al. (2004) and for the multivariate performance measures model of Joachims (2005). We now briefly explain the details for Joachims' SVM model for multivariate performance measures. Similar ideas can be applied to the structured output models of Tsochantaridis et al. (2004).

Consider the classification problem of section 2. To make the presentation short, we first consider the case of linear kernels and use the same notations as in Joachims (2005). Let $\bar{\mathcal{Y}} = \{-1, +1\}^n$, $\bar{\boldsymbol{x}} = (x_1, \ldots, x_n)$, $\bar{y} = (y_1, \ldots, y_n)$, $\bar{y}' \in \bar{\mathcal{Y}}$, and $\Psi(\bar{\boldsymbol{x}}, \bar{y}') = \sum_i y_i' x_i$. The primal problem (Optimization Problem 2 of Joachims (2005)) is

$$\begin{aligned} \min_{w,\xi} \quad & \tfrac{1}{2}\|w\|^2 + C\xi \\ s.t. \quad & \forall \bar{y}' \in \bar{\mathcal{Y}} : w^T[\Psi(\bar{\boldsymbol{x}}, \bar{y}) - \Psi(\bar{\boldsymbol{x}}, \bar{y}')] \geq \Delta(\bar{y}, \bar{y}') - \xi \end{aligned} \tag{63}$$

Although the number of inequalities is exponential, application of Algorithm 1 of Joachims (2005) usually leads to only a small number of active inequalities. We can write $[\Psi(\bar{\boldsymbol{x}}, \bar{y}) - \Psi(\bar{\boldsymbol{x}}, \bar{y}')] = Xz$ where X is a matrix that has x_i, $i = 1, \ldots, n$ as the columns and $z = \bar{y} - \bar{y}'$ is a sparse vector. With α denoting the Lagrange multiplier vector corresponding to the active inequality constraints, the optimality conditions can be written as

$$Z^T X^T X Z \alpha - \xi e - D = 0, \quad -e^T \alpha + C = 0 \tag{64}$$

where Z is a matrix that contains the z vectors of the active inequalities in its columns, e is a vector having all 1's, and D is a vector containing the $\Delta(\bar{y}, \bar{y}')$'s of the active inequalities. This is the optimality system that is parallel to (12). For nonlinear kernels we just need to replace $X^T X$ by the kernel matrix. For efficiency reasons it is useful to note that we only need to form the kernel sub matrix corresponding to the non-zero rows of Z. This model can be particularly very effective when combined with the validation objective functions of section 5.

Semi-supervised Models. There has been significant interest in semi-supervised learning recently. Most approaches extend the SVM objective function with loss terms over unlabeled data and/or use special kernel functions. For example, in the manifold regularization approach of Belkin et al. (2006), one solves optimization problems of the following form:

$$\min_{w,\xi} \quad \tfrac{1}{2}\|w\|^2 + C\sum_{i=1}^{l} l(o_i, y_i) + C^\star \sum_{i,j=1}^{n} (o_i - o_j)^2 W_{ij}$$

where only l of the n examples are labeled, W is the adjacency matrix of a data-similarity graph, and $C^\star$ is an additional hyperparameter. The last term in the objective function is a measure of how much outputs vary over similar examples. This biases learning towards weight vectors that produce smooth outputs over data clusters or manifolds. For common choices of l, one can interpret this approach as a standard SVM with a modified kernel derived in Sindhwani et al. (2006), with optimality conditions of the form in (5). Thus, one can tune $C^\star$ together with C and other kernel parameters using the gradient procedure described in section 3. An interesting direction is the design of validation functions using unlabeled data.

References

M. Belkin, P. Niyogi, and V. Sindhwani. Manifold Regularization: A Geometric Framework for Learning from Labeled and Unlabeled Examples. JMLR 2006 (to appear)

O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46:131–159, 2002.

W. Chu, C. J. Ong and S. S. Keerthi. An improved conjugate gradient scheme to the solution of least squares SVMs. *IEEE Transactions on Neural Networks*, 2005.

W. Chu and S. S. Keerthi. New approaches to support vector ordinal regression. ICML, 2005.

K. M. Chung, W. C. Kao, C. L. Sun, L. L. Wang and C. J. Lin. Radius margin bounds for support vector machines with the RBF kernel. *Neural Computation*, 15:2643–2681, 2003.

R. Collobert, S. Bengio and Y. Bengio. A parallel mixture of SVMs for very large scale problems. *Neural Computation*, 14:1105–1114, 2002.

K. Crammer and Y. Singer. On the learnability and the design of output codes for multiclass problems. COLT, 2000.

Y. Grandvalet, J. Mariéthoz and S. Bengio. A probabilistic interpretation of SVMs with an application to unbalanced classification. NIPS, 2005.

R. Herbrich, T. Graepel and K. Obermayer. Large margin rank boundaries for ordinal regression. NIPS, 2000.

A. Herschtal & B. Raskutti. Optimising Area Under the ROC Curve Using Gradient Descent ICML, 2004.

T. Joachims. A Support Vector Method for Multivariate Performance Measures ICML, 2005.

S. S. Keerthi. Efficient tuning of SVM hyperparameters using radius/margin bound and iterative algorithms. *IEEE Transactions on Neural Networks*, 13:1225–1229, 2002.

S. S. Keerthi and D. DeCoste. A modified finite Newton method for fast solution of large scale linear SVMs. JMLR, 6:341–361, 2005.

K. M. Lin and C. J. Lin. RSVM: Reduced support vector machines. SIAM Conference on Data Mining, 2001.

H. T. Lin, C. J. Lin, and R. C. Weng. A note on Platt's probabilistic outputs for support vector machines. Technical report, National Taiwan University, 2003. www.csie.ntu.edu.tw/˜cjlin/plattprob.ps.

K. M. Lin and C. J. Lin. A study on reduced support vector machines. *IEEE Transactions on Neural Networks*, 14:1449–1459, 2003.

M. Opper and O. Winther. Gaussian processes and SVM: Mean field and leave-one-out. In *Advances in Large Margin Classifiers*, pages 311–326. MIT Press, Cambridge, Massachussetts, 2000.

J. Platt. Probabilities for support vector machines. In *Advances in Large Margin Classifiers*. MIT Press, Cambridge, Massachussetts, 1999.

C. E. Rasmussen and C. K. I. Wiiliams. Gaussian Processes for Machine Learning, MIT Press, Cambridge, 2006.

R. Rifkin, G. Yeo, and T. Poggio. Regularized least squares classification. In *Advances in Learning Theory: Methods, Models and Applications*, pages 131–154. VIOS Press, Amsterdam, Nedherlands, 2003.

M. Seeger. Cross validation optimization for structured Hessian kernel methods. Tech. Report, MPI for Biological Cybernetics, Tübingen, Germany, May 2006.

A. Shashua and A. Levin. Ranking with large margin principle: Two approaches. In *Advances in Large Margin Classifiers*, pages 937–944. MIT Press, Cambridge, Massachussetts, 2003.

V. Sindhwani, P. Niyogi and M. Belkin, Beyond the point cloud: from Transductive to Semi-supervised Learning. ICML,2005.

I. Tsochantaridis, T. Hoffmann, T. Joachims and Y. Altun. Support Vector Machine learning for interdependent and structured output spaces. ICML, 2004.

J. Weston and C. Watkins. Multi-class support vector machines. ESANN, 1999.

X. Wu and R. K. Srihari. Incorporating prior knowledge with weighted margin support vector machines. KDD, 2004.

T. Zhang. Statistical behavior and consistency of classification methods based on convex risk minimization. *The Annals of Statistics*, 32:56–85, 2004.