Machine Learning, 61, 151–165, 2005
2005 Springer Science + Business Media, Inc. Manufactured in The Netherlands.
DOI: 10.1007/s10994-005-0768-5

A Fast Dual Algorithm for Kernel Logistic Regression

S.S. KEERTHI sathiya.keerthi@overture.com
Yahoo! Research Labs, 210 S. DeLacey Avenue, Pasadena, CA-91105, USA

K.B. DUAN engp9286@nus.edu.sg
Control Division, Department of Mechanical Engineering, National University of Singapore, Singapore

S.K. SHEVADE shirish@csa.iisc.ernet.in
Department of Computer Science and Automation, Indian Institute of Science, Bangalore, India

A.N. POO mpepooan@nus.edu.sg
Control Division, Department of Mechanical Engineering, National University of Singapore, Singapore

Editor: Shai Ben-David
Published online: 11 July 2005

Abstract. This paper gives a new iterative algorithm for kernel logistic regression. It is based on the solution of a dual problem using ideas similar to those of the Sequential Minimal Optimization algorithm for Support Vector Machines. Asymptotic convergence of the algorithm is proved. Computational experiments show that the algorithm is robust and fast. The algorithmic ideas can also be used to give a fast dual algorithm for solving the optimization problem arising in the inner loop of Gaussian Process classifiers.

Keywords: classification, logistic regression, kernel methods, SMO algorithm

1. Introduction

Kernel logistic regression (kLOGREG) (Jaakkola & Haussler, 1999; Roth, 2001; Wahba, 1997; Zhu & Hastie, 2001), like Support Vector Machines (SVMs) (Vapnik, 1995) is a powerful discriminative method. It also has a direct probabilistic interpretation that makes it suited for Bayesian design. In this paper we develop a fast algorithm for kLOGREG which is very much in the spirit of the popular Sequential Minimal Optimization (SMO) algorithm (Platt, 1998; Keerthi et al., 2001) for SVMs. The algorithm does not do any matrix operations involving the kernel matrix and hence is ideal for use with large scale problems. It is also extremely easy to implement.

In this paper we focus on the two category classification problem. The multi-category problem will be addressed in a future paper. Throughout we will use x to denote the input vector of the classification problem and z to denote the feature space vector which is related to x by the transformation, $z = \varphi(x)$. As in all kernel designs, we do not assume φ to be known; all computations will be done using only the kernel function, $K(x, \hat{x}) = \varphi(x) \cdot \varphi(\hat{x})$, where "$\cdot$" denotes inner product in the z space. Let $\{(x_i, y_i)\}$ denote the training set, where

x_i is the i-th input pattern and y_i is the corresponding target value; $y_i = 1$ means x_i is in class 1 and $y_i = -1$ means x_i is in class 2. Let $z_i = \varphi(x_i)$. Kernel-based classification methods solve the following optimization problem:

$$\min_{w,b} E = \frac{1}{2}\|w\|^2 + C\sum_i g(-y_i(w \cdot z_i - b)), \tag{1.1}$$

where C is a regularization parameter that is tuned using techniques such as cross validation. For kLOGREG, g is given by:

$$g(\xi) = \log(1 + e^{\xi}). \tag{1.2}$$

It is the negative log-likelihood function associated with the probabilistic model

$$\mathrm{Prob}(y \mid x) = \frac{1}{1 + e^{-y(w\cdot\varphi(x)-b)}}. \tag{1.3}$$

Using the fact that w can be written as

$$w = \sum_i \alpha_i y_i z_i, \tag{1.4}$$

the problem (1.1) becomes a finite-dimensional convex programming problem:

$$\min_{\alpha,b} E = \frac{1}{2}\sum_i\sum_j \alpha_i\alpha_j \tilde{K}(x_i, x_j) + C\sum_i g(\xi_i), \tag{1.5}$$

where $\tilde{K}(x_i, x_j) = y_i y_j K(x_i, x_j)$ and $\xi_i = y_i b - \sum_j \alpha_j \tilde{K}(x_i, x_j)$. Roth (2001) and Zhu and Hastie (2001) solve (1.5) using Newton iterations that require the inversion of $\tilde{K}$ at each iteration. When the number of training examples is even as large as a few thousands, such methods can become very expensive. An alternative is to solve (1.5) using gradient based techniques. But such methods cannot exploit certain structures present in the problem at hand. In this paper we employ the dual formulation of the form developed by Jaakkola and Haussler (1999). This leads to the replacement of (1.5) by an alternate convex programming problem[1] with a structure that is very similar to the dual arising in SVMs. This allows us to easily adapt the SMO algorithm for SVMs (Platt, 1998; Keerthi et al., 2001), which optimizes only two α_i's at each iteration (and therefore extremely easy to implement) and is known to scale efficiently to large scale problems.

The optimization problem in (1.1) (with b omitted) also occurs in the inner loop of Gaussian Process (GP) classifiers. Williams and Barber (1998) mention that *computational methods used to speed up the quadratic programming problem for SVMs may also be useful for the GP classifier problems.* Our algorithm precisely achieves that objective. For the simplified problem, our algorithm can be viewed as an improved version of the coordinate-wise descent method suggested by Jaakkola and Haussler (1999).

The paper is organized as follows. In Section 2 we develop a dual of (1.1). Optimality conditions for the dual are derived in Section 3. The ideas here form the basis for the SMO algorithm for kLOGREG developed in Section 4. In that Section we also prove that the algorithm is asymptotically convergent. Some practical aspects of the algorithm are discussed in Section 5. Computational experiments comparing the SMO algorithm for kLOGREG with the quasi-Newton method are reported in Section 6. Some concluding remarks are made in Section 7.

2. Dual formulation

To derive a dual of (1.1), we use ideas very close to those given by Cauwenberghs (2001). The optimization problem (1.1) can be rewritten as:

$$\min_{w,b} \quad E = \frac{1}{2}\|w\|^2 + C\sum_i g(\xi_i) \tag{2.1a}$$

$$\text{subject to}: \quad \xi_i = -y_i(w \cdot z_i - b) \quad \forall\, i. \tag{2.1b}$$

The Lagrangian for this problem is:

$$L = \frac{1}{2}\|w\|^2 + C\sum_i g(\xi_i) + \sum_i \alpha_i[-\xi_i - y_i(w \cdot z_i - b)].$$

The optimality conditions are given by:

$$\nabla_w L = w - \sum_i \alpha_i y_i z_i = 0 \tag{2.2a}$$

$$\frac{\partial L}{\partial b} = \sum_i \alpha_i y_i = 0 \tag{2.2b}$$

$$\frac{\partial L}{\partial \xi_i} = Cg'(\xi_i) - \alpha_i = 0 \quad \forall\, i. \tag{2.2c}$$

Note that w and ξ_i can be expressed as functions of the α_i's using (2.2a) and (2.2c):

$$w(\alpha) = \sum_i \alpha_i y_i z_i, \qquad \xi_i(\alpha_i) = g'^{-1}\left(\frac{\alpha_i}{C}\right). \tag{2.3}$$

Let $\delta = \frac{\alpha_i}{C}$. Since ξ_i can be expressed in terms of α_i, consider the function

$$G(\delta) = \delta\xi_i - g(\xi_i). \tag{2.4}$$

Note that this function forms a part of L. Differentiating G with respect to δ and using (2.2c), we get

$$\frac{dG}{d\delta} = \delta\frac{d\xi_i}{d\delta} + \xi_i - g'(\xi_i)\frac{d\xi_i}{d\delta} = \xi_i = g'^{-1}(\delta). \tag{2.5}$$

Therefore, G can be obtained using

$$G'(\delta) = g'^{-1}(\delta). \tag{2.6}$$

Remark 1. When g is given by (1.2), we have $g'(\xi) = e^{\xi}/(1 + e^{\xi})$. Thus g' is invertible. The range of g' is the open interval, (0, 1) and hence g'^{-1} is a well-defined function in the domain (0, 1).

It is easy to verify, by checking the non-negativity of second order derivatives, that, if g is a convex function then G is also a convex function. For the case of logistic regression g is given by (1.2) and we have:

$$\begin{aligned} g'^{-1}(u) &= \log(u/(1-u)) \\ G(\delta) &= \delta\log\delta + (1-\delta)\log(1-\delta) \\ G'(\delta) &= \log\left(\frac{\delta}{1-\delta}\right), \qquad G''(\delta) = \frac{1}{\delta(1-\delta)}. \end{aligned} \tag{2.7}$$

Let us now apply Wolfe duality theory to (2.1). The Wolfe dual corresponds to the maximization of L subject to (2.2a)–(2.2c), with w, b, ξ_i's and α_i's as variables. Using (2.2b), (2.3) and (2.4) we can simplify the Wolfe dual as

$$\begin{aligned} \min_{\alpha} \quad & f(\alpha) = \frac{1}{2}\|w(\alpha)\|^2 + C\sum_i G\left(\frac{\alpha_i}{C}\right) \\ \text{subject to} \quad & \sum_i \alpha_i y_i = 0. \end{aligned} \tag{2.8}$$

This is a convex programming problem. Once the α_i's are obtained by solving (2.8), the primal variables, w and ξ_i's can be determined using (2.3). The determination of b will be addressed in the next section.

3. Optimality conditions for dual

To derive proper stopping conditions for algorithms which solve the dual and also determine the threshold parameter b, it is important to write down the optimality conditions for the dual. The Lagrangian for (2.8) is:

$$\bar{L} = f - \beta\sum_i \alpha_i y_i = \frac{1}{2}\|w(\alpha)\|^2 + C\sum_i G\left(\frac{\alpha_i}{C}\right) - \beta\sum_i \alpha_i y_i. \tag{3.1}$$

Define

$$F_i = w(\alpha) \cdot z_i = \sum_j \alpha_j y_j K(x_i, x_j)$$

$$\text{and} \quad H_i = F_i + y_i G'\left(\frac{\alpha_i}{C}\right). \tag{3.2}$$

The optimality conditions for the dual problem are:

$$\frac{\partial \bar{L}}{\partial \alpha_i} = (H_i - \beta) y_i = 0 \quad \forall\, i. \tag{3.3}$$

Define:

$$b_{\text{up}} = \max_i H_i \qquad i_{\text{up}} = \arg\max_i H_i \tag{3.4a}$$

$$b_{\text{low}} = \min_i H_i \qquad i_{\text{low}} = \arg\min_i H_i. \tag{3.4b}$$

Then optimality conditions will hold at a given α iff

$$b_{\text{low}} = b_{\text{up}}. \tag{3.5}$$

Remark 2. In the above discussion, note that f, F_i, H_i, b_{up}, i_{up}, b_{low} and i_{low} are all functions of α. The functional dependancies have not been put down to avoid notational clutter. These functions are appropriately defined on some set A in the α space; for instance, in the case of g given by (1.2), Remark 1 implies that

$$A = \{\alpha : 0 < \alpha_i < C \quad \forall i\ \}. \tag{3.6}$$

Using (3.3), (3.2), (2.5) and (2.1b), it is easy to see the close relationship between the threshold parameter b in the primal problem and the multiplier, β. *In particular, at optimality, β and b are identical.* Therefore, in the rest of the paper β and b will denote one and the same quantity.

We will say that an index pair (i, j) defines a *violation* at α if

$$H_i \neq H_j. \tag{3.7}$$

Thus, optimality conditions will hold at α iff there does not exist any index pair (i, j) that defines a violation.

Suppose (i, j) satisfies (3.7) at some α. Then it is possible to achieve a decrease in f (while maintaining the equality constraint, $\sum \alpha_k y_k = 0$) by adjusting α_i and α_j only. To see this, let us define the following:

$$\begin{aligned} &\tilde{\alpha}_i(t) = \alpha_i + t/y_i, \quad \tilde{\alpha}_j(t) = \alpha_j - t/y_j, \\ &\tilde{\alpha}_k(t) = \alpha_k \quad \forall k \neq i, j, \end{aligned} \tag{3.8}$$

and

$$\phi(t) = f(\tilde{\alpha}(t)). \tag{3.9}$$

For logistic regression f is strictly convex, and hence, by (3.8) and (3.9) ϕ is also strictly convex. The domain of ϕ is the open interval defined by $\{t : \tilde{\alpha}(t) \in A\}$. It is easy to verify that

$$\phi'(t) = H_i - H_j, \tag{3.10}$$

where H_i and H_j are evaluated at $\tilde{\alpha}(t)$. Since, by (3.7), $H_i - H_j \neq 0$ at $t = 0$, a decrease in ϕ is possible by choosing t suitably away from 0.

Since, in numerical solution, it is usually not possible to achieve optimality exactly, there is a need to define approximate optimality conditions. The condition (3.5) can be replaced by

$$b_{\text{low}} \geq b_{\text{up}} - 2\tau, \tag{3.11}$$

where τ is a positive tolerance parameter. Once (3.11) is achieved, we can take

$$b = \frac{b_{\text{low}} + b_{\text{up}}}{2} \tag{3.12}$$

for use with (1.3).

A useful alternative for stopping and choosing threshold is to employ the duality gap, $D_{\text{gap}} = E + f$. By Wolfe duality theory: D_{gap} is nonnegative; and, $D_{\text{gap}} = 0$ iff optimality holds. Thus we can use the stopping criterion:

$$D_{\text{gap}} \leq \epsilon |f|, \tag{3.13}$$

where ϵ is a suitable positive tolerance. D_{gap} can be computed as follows. Given α, let $w(\alpha)$ be given by (2.3) and $\xi(b)$ be obtained from (2.1b). Then

$$D_{\text{gap}} = E + f = \|w(\alpha)\|^2 + C \sum_i \left[G\left(\frac{\alpha_i}{C}\right) + g(\xi_i(b)) \right]. \tag{3.14}$$

Also, b can be chosen to minimize D_{gap}. This is equivalent to minimizing $\sum_i g(\xi_i(b))$, which can be numerically done using Newton-Raphson iterations.

4. SMO algorithm for kLOGREG

In this section we give the SMO algorithm for kLOGREG, for which g is given by (1.2). A basic step consists of starting with a point α and optimizing only two variables α_i and α_j to form the new point α_{new}. Consider (3.8) and (3.9). Given (3.10), the natural choice is $i = i_{\text{up}}$ and $j = i_{\text{low}}$ so as to make $|\phi'(0)|$ as large as possible. Using the notations of Section 3, we can write the optimization problem and the resulting solution as

$$t^\star = \arg\min_t \phi(t) \quad \text{and} \quad \alpha_{\text{new}} = \tilde{\alpha}(t^\star). \tag{4.1}$$

The SMO algorithm can now be described.

SMO Algorithm for kLOGREG.

1. Choose $\alpha^0 \in A$ and set $r = 0$.
2. If α^r satisfies (3.5), stop. If not, set $\alpha = \alpha^r$, choose $i = i_{\text{up}}$, $j = i_{\text{low}}$ and solve (4.1).
3. Let $\alpha^{r+1} = \alpha_{\text{new}}$, $r := r + 1$ and go back to step 2.

Remark 3. Let $B = \{\alpha \in A : f(\alpha) \leq f(\alpha^0)\}$. Continuity of f implies that B is closed in A; since $B \subset A$, B is also bounded; furthermore, since $f(\hat{\alpha}^k) \to \infty$ for any sequence $\hat{\alpha}^k$ that goes to a boundary point of A, B is compact in R^m (m is the number of α variables, which is same as the number of training examples). Since the SMO algorithm is a descent algorithm, $\alpha^r \in B\ \forall r$. Thus every iteration of the algorithm is well-defined.

We now establish the convergence of the SMO algorithm described above. The absence of 'hard boundaries' in the optimization makes the proof of convergence much simpler than corresponding proofs for SMO algorithm for SVMs. We first establish a useful result.

Lemma 1. *The following holds for each* $r \geq 0$:

$$f(\alpha^r) - f(\alpha^{r+1}) \geq \frac{2}{C}\|\alpha^{r+1} - \alpha^r\|^2.$$

Proof: The second order truncated Taylor series expansion of ϕ around $t^\star$ is given by

$$\phi(t) = \phi(t^\star) + \frac{1}{2}\phi''(\tilde{t})(t - t^\star)^2, \tag{4.2}$$

where $\tilde{t}$ lies in between t and $t^\star$ and is dependent on them. The second order derivative of ϕ has the expression

$$\phi''(t) = \eta + \frac{1}{C}\left[G''\left(\frac{\tilde{\alpha}_i(t)}{C}\right) + G''\left(\frac{\tilde{\alpha}_j(t)}{C}\right)\right], \tag{4.3}$$

where $\eta = K(x_i, x_i) - 2K(x_i, x_j) + K(x_j, x_j)$. Using the expression for G'' in (2.7) we can get the bound, $\phi''(t) \geq (8/C)$. Employing this in (4.2) and setting $t = 0$ we get

$$f(\alpha^r) - f(\alpha^{r+1}) = \phi(0) - \phi(t^\star) \geq \frac{4}{C}(t^\star)^2 = \frac{2}{C}\|\alpha^{r+1} - \alpha^r\|^2.$$

This proves Lemma 1. □

Theorem 1. *The following hold.*

1. $\{\alpha^r\}$ *has at least one limit point in* A.
2. *Every limit point of* $\{\alpha^r\}$ *lies in* A *and it is a solution of* (2.8).

Proof: By Remark 3 $\{\alpha^r\} \subset B$. Since B is compact in R^m, $\{\alpha^r\}$ has a limit point in B; also, every limit point of $\{\alpha^r\}$ lies in B. Since $B \subset A$, every limit point of $\{\alpha^r\}$ lies in A too.

Since the algorithm decreases f at each step and f is bounded below, $\{f(\alpha^r)\}$ is a convergent sequence. By Lemma 1 we immediately get that $\{\alpha^{r+1} - \alpha^r\}$ converges to 0.

Now let $\{\alpha^{r(s)}\}_{s\geq 0}$ denote a convergent subsequence and $\bar{\alpha}$ denote the limit point in A to which it converges. For any $r \geq 0$, let $i(r) = i_{\text{up}}(\alpha^r)$ and $j(r) = i_{\text{low}}(\alpha^r)$, the two indices chosen for optimization at the r-th step. Since $\phi'(t^\star) = 0$ for $t^\star$ given by (4.1), we get from (3.10) that

$$H_{i(r)}(\alpha^{r+1}) - H_{j(r)}(\alpha^{r+1}) = 0. \tag{4.4}$$

Since there are only a finite number of indices, there exists at least one pair (i_1, j_1) such that $i_1 = i(r(s))$ and $j_1 = j(r(s))$ for infinitely many s. Let us restrict ourselves to only such a subsequence. To keep notations simple, let us rename the subsequence and take that

$$\begin{aligned} i_1 &= i(r(s)) = i_{\text{up}}\big(\alpha^{r(s)}\big) \quad \text{and} \\ j_1 &= j(r(s)) = i_{\text{low}}\big(\alpha^{r(s)}\big) \quad \forall\, s \geq 0. \end{aligned}$$

Since b_{up} and b_{low} are continuous functions of α, we also get

$$\begin{aligned} b_{\text{up}}(\bar{\alpha}) - b_{\text{low}}(\bar{\alpha}) &= \lim_{s\to\infty} \big[b_{\text{up}}\big(\alpha^{r(s)}\big) - b_{\text{low}}\big(\alpha^{r(s)}\big)\big] \\ &= \lim_{s\to\infty} \big[H_{i_1}\big(\alpha^{r(s)}\big) - H_{j_1}\big(\alpha^{r(s)}\big)\big] \\ &= \lim_{s\to\infty} [P(s) + Q(s) + R(s)], \end{aligned} \tag{4.5}$$

where

$$\begin{aligned} P(s) &= \big[H_{i_1}\big(\alpha^{r(s)}\big) - H_{i_1}\big(\alpha^{r(s)+1}\big)\big] \\ Q(s) &= \big[H_{i_1}\big(\alpha^{r(s)+1}\big) - H_{j_1}\big(\alpha^{r(s)+1}\big)\big] \\ R(s) &= \big[H_{j_1}\big(\alpha^{r(s)+1}\big) - H_{j_1}\big(\alpha^{r(s)}\big)\big]. \end{aligned} \tag{4.6}$$

Since $\{\alpha^{r(s)+1} - \alpha^{r(s)}\}$ converges to 0, $\lim_{s\to\infty} P(s) = 0$ and $\lim_{s\to\infty} R(s) = 0$. By (4.4), $Q(s) = 0 \;\; \forall s$. Thus (4.5) yields $b_{\text{up}}(\bar{\alpha}) - b_{\text{low}}(\bar{\alpha}) = 0$. By (3.5), $\bar{\alpha}$ is a solution of (2.8). This completes the proof. □

5. Practical aspects

In practice, we can use (3.11) instead of (3.5) in step 2 of the SMO algorithm. When this is done, one expects the algorithm to converge to an approximate solution satisfying (3.11) within a finite number of steps.

The univariate optimization problem (4.1) can be solved using Newton-Raphson iterations:

$$t^{l+1} = t^l - [\phi''(t^l)]^{-1}\phi'(t^l) \tag{5.1}$$

starting from $t^0 = 0$ and until a certain accuracy is reached. (To get guaranteed convergence, we can suitably combine Newton-Raphson iterations with some bisection steps when necessary.) With the required accuracy (3.11) in mind, we can terminate the iterations in (5.1) when we find a point t^l satisfying a tighter accuracy criterion, say $\phi'(t^l) < 0.1\tau$. While $\phi'(t^l)$ is given by (3.10), $\phi''(t^l)$ can be computed using the formula (4.3).

Since the function H_k plays an important role in the algorithm it is better to maintain a cache for $\{H_k\}$. At the end of the k-th step involving indices i and j, we can use the update formula

$$\begin{aligned} H_k(\alpha^{r+1}) &= H_k(\alpha^r) + y_i\big[\alpha_i^{r+1} - \alpha_i^r\big]K(x_k, x_i) \\ &\quad + \ y_j\big[\alpha_j^{r+1} - \alpha_j^r\big]K(x_k, x_j) \\ &= H_k(\alpha^r) + t^\star[K(x_k, x_i) - K(x_k, x_j)] \quad \forall\, k \neq i, j. \end{aligned} \tag{5.2}$$

For $k = i, j$, $H_k(\tilde{\alpha}(t))$ is needed at various t^l values in order to implement (5.1) via (3.10). For these two special indices, we can use the following update formula:

$$\begin{aligned} H_k(\tilde{\alpha}(t^{l+1})) = {} & H_k(\tilde{\alpha}(t^l)) \\ & + y_i(\tilde{\alpha}_i(t^{l+1}) - \tilde{\alpha}_i(t^l))K(x_k, x_i) \\ & + y_j(\tilde{\alpha}_j(t^{l+1}) - \tilde{\alpha}_j(t^l))K(x_k, x_j) \\ & + y_k\left[G'\left(\frac{\tilde{\alpha}_k(t^{l+1})}{C}\right) - G'\left(\frac{\tilde{\alpha}_k(t^l)}{C}\right)\right] \quad \text{for} \quad k = i, j. \end{aligned} \tag{5.3}$$

At each step, the solution of (4.1) via (5.1), (3.10), (5.3) and (4.3) is very efficient and takes very little (constant time) effort. The updating of H_k by (5.2) after completion of the solution of (4.1) requires $\mathcal{O}(m)$ effort where m is the number of training examples; it forms the main bulk of the computational cost.

The solution of (4.1) can come across a certain ill-conditioned situation which requires special handling. Let $\tilde{\alpha}(t)$ be as in (3.8). From (3.10) and (3.2) we have

$$\begin{aligned}0 &= \phi'(t^\star) = H_i - H_j \\ &= F_i - F_j + y_i G'\left(\frac{\tilde{\alpha}_i(t^\star)}{C}\right) - y_j G'\left(\frac{\tilde{\alpha}_j(t^\star)}{C}\right).\end{aligned}$$

Suppose the size of $F_i - F_j$ is in the order of 10^5. (Such sizes are very much possible when a large value is tried for C.) Therefore, for $H_i - H_j = 0$ to occur, we require the size of $G'(\frac{\tilde{\alpha}_i(t^\star)}{C})$ and/or $G'(\frac{\tilde{\alpha}_j(t^\star)}{C})$ to be in the order of 10^5, which is possible only if at least one of $\tilde{\alpha}_i(t^\star)$, $C - \tilde{\alpha}_i(t^\star)$, $\tilde{\alpha}_j(t^\star)$, or $C - \tilde{\alpha}_j(t^\star)$ is extremely small, i.e., with size e^{-10^5}. In such a case, a reliable determination of $t^\star$ is messy and difficult. As we now explain, an accurate determination of $t^\star$ in this case is actually unnecessary. Suppose $t^\star$, the solution of (4.1) is such that one of the variables, say $\tilde{\alpha}_i(t^\star)$ is extremely close to 0 or C. Since pushing $\tilde{\alpha}_i(t)$ to an accurate value close to 0 or C has only to do with setting $y_i G'(\frac{\tilde{\alpha}_i(t)}{C})$ precisely and it has little effect on F_i or F_j, the accurate determination of $t^\star$ is unimportant. However, having said that, we should also note that, if we decide to avoid a precise determination of $t^\star$ then the value of H_i becomes unreliable and so such indices have to be treated specially when checking for optimality.

To handle the issue cleanly and reliably, we proceed as follows. Let μ be a small number, say $10^3\times$ *machine precision*. Define $I = (0, C)$ and $\tilde{I} = (\mu C, C - \mu C)$. If, during the solution of (4.1), we come across a situation[2] at which we know that for an index, say i, we have $\alpha_i(t^\star) \in I \backslash \tilde{I}$, then we terminate the solution of (4.1) and place $\tilde{\alpha}_i(t)$ at the appropriate end point of $\tilde{I}$ (i.e., μC or $C - \mu C$). In that case, since H_i is unreliable we need to treat such indices specially. So we put such indices in a special group called NBG (Near Boundary group). Other indices whose α values lie inside $\tilde{I}$ will be put in NG (Normal group).

Once an index gets into NBG it is best not to involve it in further optimization. However, at the end of the optimization, a check on indices in NBG has to be conducted to be sure that moving such indices back to NG does not lead to an improvement in objective function. Thus a two loop approach is needed for the SMO algorithm.[3] *Since H_i, $i \in$ NBG are not reliable, at any stage of the algorithm we always compute i_{up}, b_{up}, i_{low} and b_{low} using only indices from* NG. The inner loop repeatedly operates steps 2 and 3 of the SMO algorithm, using (3.11) instead of (3.5) so as to obtain finite termination. When the inner loop satisfies (3.11), we go into the outer loop where each index, $i \in$ NBG is checked for optimality. This is done by attempting to solve (4.1) twice, once with $j = i_{\text{low}}$ and then again with $j = i_{\text{up}}$. If, in each of these solutions we find that no change has occured (i.e., $i \in$ NBG and α_i remains at the same end point of $\tilde{I}$), then optimality holds as far as i is concerned. If, during the outer loop, α_i changes even for one i, then the inner loop is entered again after the outer loop is completed. On the other hand, if none of the α_i has changed in the outer loop, then optimality holds for all i and the SMO algorithm is terminated.

6. Numerical experiments

First we empirically evaluate the computational cost of our SMO algorithm for kLOGREG. Note that this algorithm solves the dual (2.8) and that the corresponding primal formulation

(2.1) is equivalent to the formulation (1.5). To solve kLOGREG, Roth (2001) and Zhu and Hastie (2001) used second order optimization methods that require the storage and inversion of the Hessian matrix, to solve (1.5). When the number of examples is more than a few thousand these methods become very expensive. So, with the solution of larger problems also in mind, we compare our method with the limited memory BFGS algorithm[4] (Liu and Nocedal) for solving (1.5). Since our algorithm solves the dual and the BFGS algorithm solves the primal and they use different approximate stopping criteria, comparison of their computational costs becomes difficult. To make the comparison fair, we proceed as follows. First we solve the dual by our SMO algorithm using (3.11) for stopping, and note the computing time required. The α, along with the value of b (see (3.12)) obtained by the SMO algorithm are used to define a feasible (w, b) for the primal problem (1.1). This (w, b) attains a certain (sub-optimal) value for the primal objective function. The BFGS algorithm for solving (1.5) is then run until the above value of the primal objective function is reached. The corresponding computing time was used for comparison purposes.

The SMO algorithm for kLOGREG was implemented in C and executed on the Sun Blade 100 workstation which uses a 500 MHz UltraSPARC-IIe processor and the Solaris OS. For the BFGS method, the freely available software at the site `http://www.ece.nwu.edu/~nocedal/lbfgs.html` was used. The Gaussian kernel $K(x, \tilde{x}) = \exp(-\frac{\|x-\tilde{x}\|^2}{2\sigma^2})$ was used. In all the experiments, τ was set to 10^{-6}. Five benchmark datasets were used: Banana, Image, Splice, Waveform and Tree. The Tree dataset was originally used in Bailey et al. (1993). Detailed information about the remaining datasets can be found in Rätsch (1999). Some details about these datasets are given in Table 1.

Let us now explain how the α's were initialized. For the SMO algorithm it is necessary to have $\alpha_i \in (0, C)\ \forall i$. Let m_1 and m_2 denote, respectively, the number of training examples in class 1 and class 2. The α's were initialized to $\frac{C}{m_1}$ and $\frac{C}{m_2}$ respectively for the examples in class 1 and class 2. This initialization was used for both the SMO algorithm as well as the BFGS algorithm. Unlike the SMO algorithm, the BFGS algorithm for (1.5) can actually be initialized with any values for the α's. However, it was observed that there was no noticeable change in the CPU times for the BFGS algorithm when the α's were initialized to values other than those mentioned above, for example setting all α's to zero.

Just for the purpose of comparing training times σ was fixed at a specific value which is optimal for the generalization performance of kLOGREG. The CPU times for different

Table 1. Properties of datasets.

Dataset	Number of input variables	Number of training examples	Number of test examples
Banana	2	400	4900
Splice	60	1000	2175
Waveform	21	400	4600
Tree	18	700	11692
Image	18	1300	1010

Table 2. Computational costs for SMO and BFGS algorithms.

	Banana $\sigma^2 = 0.4297$		Splice $\sigma^2 = 43.8856$		Waveform $\sigma^2 = 15.2735$		Tree $\sigma^2 = 2.00$		Image $\sigma^2 = 1.3776$	
$\tilde{C}$	SMO	BFGS	SMO	BFGS	SMO	BFGS	SMO	BFGS	SMO	BFGS
−4	0.6	23.0	18.0	1.1e3	2.4	69.6	3.9	200.1	40.6	916.4
−3	0.6	15.2	16.7	588.2	2.1	42.4	3.4	153.4	41.2	520.2
−2	0.5	13.4	14.0	760.1	2.1	57.2	2.5	217.8	31.1	671.0
−1	0.3	55.4	10.2	2.3e3	1.5	156.1	2.6	1.1e3	25.5	3.2e3
0	0.5	255.2	13.2	6.1e3	2.9	478.8	4.0	5.5e3	41.0	9.2e3
1	1.2	963.1	22.1	2.88e4	5.5	1.9e3	7.2	4.3e4	63.2	4.6e4
2	4.0	3.1e3	32.0	–	13.0	3.5e3	20.7	–	99.0	–
3	41.6	–	40.0	–	20.5	5.8e3	1.1e2	–	178.6	–
4	8.4e2	–	54.2	–	24.9	–	7.1e2	–	6.2e2	–

The first column indicates $\tilde{C}$, which is $\log_{10} C$. Each unit denotes CPU time (in seconds). "–" denotes the cases for which CPU times were larger than 50000 seconds and hence training was abandoned.

datasets are given in Table 2 as functions of C. From this table it is clear that the SMO algorithm for kLOGREG is very much faster than the BFGS algorithm. The difference is much higher for large values of C.

To see how the cost of the SMO algorithm scales with data size, an experiment was done on the UCI "Adult" dataset (Merz and Murphy, 1998) by gradually increasing the training set size from 1605 to 22696 in eight steps and observing the training time. A line was then fitted to the plot of the log of the training time versus the log of the training set size. The slope of this line is the empirical scaling exponent. The datasets of different sizes that are used are available in http://www.research.microsoft.com/˜jplatt/adult.zip. The training was done with both, the linear kernel ($C = 0.05$) and the Gaussian kernel ($C = 1.0$ and $\sigma^2 = 10$). The SMO algorithm for kLOGREG scales well on this dataset, with the scaling exponent of 2.2 on both, the linear kernel as well as the Gaussian kernel; thus computing time is roughly proportional to $m^{2.2}$ where m is the training set size.

Kernel logistic regression minimizes the negative log-likelihood function associated with a probabilistic model along with the regularizer term. Thus it naturally provides values for posterior class probabilities. To see how good the designed probabilistic model is, we first compared it with the optimal Bayes classifier on an artificial two-category classification problem. For this purpose, the examples in the two classes were generated using Gaussian distributions with the following means and covariance matrices: $\mu_1 = (-2, 0)$, $\Lambda_1 = \text{Diag}\{1, 2\}$, $\mu_2 = (2, 0)$, $\Lambda_2 = \text{Diag}\{2, 1\}$. The priors for the two classes were taken to be equal. 400 training points were used. A test set of size 20000 was generated using the same distributions.

Five-fold cross validation was used to tune the hyperparameters involved in the problem formulations (that is, C and σ) and the test set error was obtained using the optimal hyperparameter values for each of the formulations. The initial search for optimal hyperparameters

Table 3. Negative log-likelihood of the test set (NLL) and the fraction of test set errors (TErr) for optimal Bayes classifier (Bayes), kLOGREG (KLR) and SVM on the two dimensional artificial dataset.

Method	NLL	TErr
Bayes	2532.5	0.0490
KLR	2663.4	0.0502
SVM	2703.5	0.0507

Table 4. Generalization performance comparison of kLOGREG (KLR) and SVM on the five benchmark datasets.

	NLL		TErr	
Dataset	KLR	SVM	KLR	SVM
Banana	1328.44	1378.39	.1245	.1247
Image	85.12	83.26	.0178	.0198
Splice	615.22	542.61	.0952	.0989
Waveform	1162.93	1137.70	.1041	.1063
Tree	3547.15	3116.32	.1129	.1123

was done on a 10×10 uniform coarse grid in the $(\log C, \log \sigma)$ space, followed by a fine search on a 20×20 uniform grid in the (C, σ) space placed around the optimal pair found by the coarse search.

Table 3 gives the negative log-likelihood of the test set and the fraction of test errors for the optimal Bayes classifier and the kLOGREG method. This table also gives the corresponding values for SVM with posterior probabilities assigned in a post-processing step (Platt, 1999). Clearly, both kLOGREG and SVM perform quite well.

To further study and compare the generalization capabilities of kLOGREG and SVM methods, we determined their performance on the five benchmark datasets mentioned earlier. As in the artificial dataset, five fold cross validation was used to tune the hyperparameters C and σ. The test set results are given in Table 4. It is clear that the generalization capabilities of both methods are comparable. This observation is consistent with that made by Platt (1999).

7. Conclusion

In this paper we have given a new algorithm for kLOGREG, proved its convergence and discussed implementation aspects. The algorithm solves the dual problem. It is very much faster than the BFGS algorithm applied to the primal problem. The algorithm scales nicely to large size problems. It is also robust in the sense that on many complex datasets we have tried there was not even a single case of failure. The generalization performance of kLOGREG is comparable to that of SVMs.

The in-built probabilistic model makes kLOGREG suitable for use with Bayesian design. In fact, the algorithmic ideas given in this paper can be easily adapted for solving the optimization problem arising in the inner loop of Gaussian Process classifiers. This optimization problem is simpler to solve than (1.1) since b is absent, thereby getting rid of the equality constraint in (2.8).[5] Correspondingly, the optimality conditions in (3.3) get replaced by the following conditions:

$$H_i = 0 \quad \forall\, i. \tag{7.1}$$

Because of the absence of an equality constraint on α, it is possible to decrease f by minimizing only one α_i at a time. Jaakkola and Haussler (1999) suggest the Gauss-Seidel coordinate-wise descent method for doing this. A better option is to choose, at any given situation, the index, $i = \arg\min_k |H_k|$ (i.e., i is the index that violates (7.1) most) and optimize α_i only. Together with a cache for $\{H_k\}$ and update formulas similar to (5.2) and (5.3), such an algorithm will be very efficient.

kLOGREG does not enjoy the sparsity property associated with SVMs. (Note that $\alpha_i \in A$ and therefore $\alpha_i > 0$ for all i.) Recent research by Zhu and Hastie (2001) has initiated useful ways of incorporating sparsity in kLOGREG. Further work along these lines, together with fast algorithms such as the one in this paper are expected to make kLOGREG an attractive tool for solving classification problems.

Acknowledgments

We thank the anonymous referees who carefully read the manuscript and gave several useful suggestions that helped us to make significant improvements to the presentation.

Notes

1. Although both, the dual problem and (1.5), involve α_i's as the variables and lead to the same solutions, their structures are markedly different.
2. This situation usually arises when the solution process of (4.1) necessarily pushes either $\tilde{\alpha}_i(t)$ or $\tilde{\alpha}_j(t)$ to a value outside $\tilde{I}$, i.e., at a t corresponding to an end point of $\tilde{I}$, descent in ϕ requires a movement out of $\tilde{I}$.
3. This is somewhat similar to what is done in the SMO algorithm for SVMs.
4. In our experiments the number of memory steps used by BFGS is 5.
5. In Gaussian Process classifiers, the effect of b can be taken care of by adding a constant to the covariance function.

References

Bailey, R. R., Pettit, E. J., Borochoff, R. T., Manry, M. T., & Jiang, X. (1993). Automatic recognition of USGS land use/cover categories using statistical and neural network classifiers. In *Proceedings of SPIE*, Vol. 1944.

Cauwenberghs, G. (2001). Kernel machine learning: A systems perspective. Tutorial presented at ISCAS. Available at http://bach.ece.jhu.edu/svm/iscas2001/iscas2001.pdf.

Jaakkola T., & Haussler, D. (1999). Probabilistic kernel regression models. In *Proceedings of the Seventh International Workshop on Artificial Intelligence and Statistics*. San Francisco: Morgan Kaufmann, See http://alphabits.ai.mit.edu/people/tommi/papers/probker.ps.gz.

Keerthi, S. S., Shevade, S. K., Bhattacharyya, C., & Murthy, K. R. K. (2001). Improvements to Platt's SMO algorithm for SVM classifier design. *Neural Computation, 13:3*, 637–649.

Merz C. J., & Murphy, P. M. (1998). UCI repository of machine learning databases. Department of Information and Computer Science, University of California, Irvine, CA. See http://www.ics.uci.edu/˜mlearn/MLRepository.html.

Platt, J. (1998). Sequential minimal optimization: A fast algorithm for training support vector machines. Tech. Rept. MSR-TR-98-14, Microsoft Research, Redmond, 1998.

Platt, J. (1999). Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In A. Smola, B. Schölkopf, & D. Schuurmans (Eds.), *Advances in large margin classifiers*. Cambridge, MA: MIT Press.

Rätsch, G. (1999). Benchmark datasets. Available at http://ida.first.gmd.de/˜raetsch/data/benchmarks.htm.

Roth, V. (2001). Probabilistic discriminative kernel classifiers for multi-class problems. In B. Radig & S. Florczyk (Eds.), *Pattern recognition–DAGM'01* (pp. 246–253). Springer, 2001. Available at http://www-dbv.informatik.uni-bonn.de/pdf/roth.dagm01.pdf.

Vapnik, V. (1995). *The Nature of Statistical Learning Theory*. New York: Springer Verlag.

Wahba, G. (1997). Support vector machines, reproducing kernel Hilbert spaces and the randomized GACV. Technical Report 984, Department of Statistics, University of Wisconsin, Madison.

Williams, C. K. I., & Barber, D. (1998). Bayesian classification with Gaussian processes. *IEEE Transactions on PAMI, 20*, 1342–1351.

Zhu, J., & T. Hastie, (2001). Kernel logistic regression and the import vector machine. In *Advances in Neural Information Processing Systems 13*. Available at http://www.stanford.edu/˜jzhu/nips01.ps.

Received July 8, 2002
Revised March 18, 2004
Accepted February 17, 2005