A Fast Tracking Algorithm for Generalized LARS/LASSO

S Sathiya Keerthi and Shirish Shevade

Abstract— **This paper gives an efficient algorithm for tracking the solution curve of sparse logistic regression with respect to the L_1 regularization parameter. The algorithm is based on approximating the logistic regression loss by a piecewise quadratic function, using Rosset and Zhu's path tracking algorithm on the approximate problem, and then applying a correction to get to the true path. Application of the algorithm to text classification and sparse kernel logistic regression shows that the algorithm is efficient.**

Index Terms— **Sparse logistic regression, Generalized LARS, LASSO**

I. Introduction

Consider a binary classification problem with parameter vector $\beta \in R^m$ and training set, $\{(x_i, t_i)\}_{i=1}^n$ where: $x_i \in R^m$ is the input vector of the i-th training example and t_i is the corresponding target taking values from $\{1, -1\}$. Using the linear model

$$y_i = \beta^T x_i \tag{1}$$

and the probability function

$$P(t_i|x_i) = \frac{1}{1+e^{-r_i}}, \quad r_i = t_i y_i \tag{2}$$

we get the training problem corresponding to L_2-regularized logistic regression as

$$\min_\beta f_0(\beta) = \frac{\mu}{2}\beta^T K \beta + \sum_{i=1}^n l(r_i) \tag{3}$$

where $l(r) = \log(1+e^{-r})$ is the logistic regression loss function and K is a symmetric positive semidefinite regularization matrix.

The logistic regression model given above has been popularly used (usually with $K = I$ where I is the identity matrix) in applications such as text categorization [4] and gene selection for microarray data [11, 12]. Kernel logistic regression (KLR) [14], which is a powerful tool for building nonlinear classifiers, also fits into our model. In KLR, we have: $m = n$, $x_{ij} = k(z_i, z_j)$, and $K_{ij} = k(z_i, z_j)$, where z_i, $i = 1, \ldots, n$, are the original training input vectors, and k is the kernel function; the effect of the bias term can be brought about by adding a constant to the kernel function. In all these mentioned applications, the number of coefficients in β [1] is large and also, a small fraction of them are sufficient for achieving the best possible classification accuracy. Sparse logistic regression is a modified model that is very effective in the selection of a relevant subset of coefficients.[2] The modification is done by including the L_1 regularizer:

$$\min_\beta f_\lambda(\beta) = f_0(\beta) + \lambda\|\beta\|_1 \tag{4}$$

This formulation which uses, both L_2 and L_1 regularizers, is called as the *Elastic Net* model [15]; as shown in [15], sometimes there is value in keeping both regularizers. The well-known LASSO model [13,4,11,12] is a special case of (4) and corresponds to setting $\mu = 0$. Throughout this paper we will consider the model in (4) and take μ to be some fixed value. Our focus is on the tracking of the solution of (4) with respect to λ. When λ is large[3] $\beta = 0$ is the minimizer of f_λ, which corresponds to the case of all coefficients being excluded. As λ is decreased, more and more coefficients take positive values. When $\lambda \to 0$, the solution of (4) approaches the minimizer of f_0, where all β_j are typically non-zero. Thus λ offers a very useful and controlled way of obtaining sparse solutions.

Recently, some fast algorithms have been given for solving (4). Genkin et al [4] and Shevade and Keerthi [12] have given a cyclic coordinate descent method that is quite efficient. Roth [11] gives an interesting variation of the IRLS (Iteratively Reweighted Least Squares) method. These algorithms are set-up to solve (4) for a given value of λ. When (4) is to be solved for several values of λ (say, during the determination of λ by cross validation) these algorithms efficiently obtain the solutions by seeding, i.e., the β obtained at one λ is used to start-off the solution at the next nearby value of λ. Even then, they are not efficient enough if a fine tracking of solutions with respect to λ is needed.

There are reasons why it is useful to have an efficient algorithm that finely tracks the solution of (4) as a function of λ.

1) Together with cross validation such an algorithm can be used to locate the best value of λ precisely.

S Sathiya Keerthi is with Yahoo! Research Labs, 210 S.DeLacey Avenue, Pasadena, CA-91105, USA. Email: selvarak@yahoo-inc.com

Shirish Shevade is with the Department of Computer Science and Automation, Indian Institute of Science, Bangalore, India. Email: shirish@csa.iisc.ernet.in

[1]We will assume that the bias term of the classifier is built into β and that, a value of 1 is put in the corresponding component of each x_i. We will also assume that each of the other coefficients is also suitably normalized, say to have unit variance; this is usually needed for the regularizer to work effectively.

[2]We are not aware of previous works which use the L_1 regularizer for sparse KLR. This method provides an interesting and useful alternative to Zou and Hastie's Import Vector Machine [14].

[3]It is easy to see that, if $\lambda > \max_j |\frac{\partial f_0}{\partial \beta_j}(0)|$, then the directional derivatives of the $\lambda\|\beta\|_1$ term at $\beta = 0$ dominate and so $\beta = 0$ is the minimizer of f_λ.

2) We can get a good understanding of the coefficients and their effects on the solution.
3) Many applications place a constraint on the number of non-zero coefficients. For example, in text categorization there may be a limit on the number of features that can be used in the final classifier for fast online processing. In KLR there may be a need to minimize the number of basis functions that define the classifier [14]. Tracking offers a direct way of enforcing such constraints.

Thus, if it is possible to derive an efficient tracking algorithm that is nearly as efficient as the algorithms mentioned earlier, it can be very useful. Developing such a tracking algorithm is the main theme of this paper.

For least squares problems, tracking solutions with respect to the λ parameter has recently become very popular. For the LASSO model, Osborne et al [7] and Efron et al [2] showed that the solution path in β space is piecewise linear and gave efficient algorithms for tracking this path. Efron et al [2] also derived the LARS algorithm which nearly yields the same path as the LASSO algorithm, but is much simpler. Rosset and Zhu [9] described a whole range of problems for which the path is piecewise linear. Keerthi [5] used their ideas to derive an effective algorithm for feature selection with linear SVMs.

For logistic regression, the literature on tracking of the solution path with respect to λ is very limited. Madigan and Ridgeway [6] provided some rudimentary ideas for tracking. Rosset [10] gave a simple algorithm for tracking by starting with a large λ at which $\beta = 0$ is the solution, varying λ in ϵ decrements, and using the β at one λ to seed the solution at $\lambda - \epsilon$. This method is slow and inadequate for large problems. Bach et al [1] consider predictor-corrector methods (for a related kernel problem); these methods require repeated factorizations of the Hessian matrix, and so they are expensive for large number of coefficients.

In this paper we derive an efficient algorithm for tracking the solution of (4) with respect to λ by first making good use of Rosset and Zhu's ideas [9] to track an approximate path and then using a pseudo-Newton correction process to get to the solution of (4); in the next two sections we give full details of these two key components of our method. In section 4 we demonstrate the efficiency of the method by making comparisons with the BBR software of Genkin et al [4].

II. Approximate tracking

We first approximate the logistic regression loss function l in (3) by a suitable $\hat{l}$ that is non-negative, convex, differentiable and, most importantly, piecewise quadratic. *This approximation is independent of the problem being solved and has to be done just once.* With such an $\hat{l}$ available, our method for a given problem comprises two main steps. In the first step, we track the solution path corresponding to $\hat{l}$ by using the ideas of Rosset and Zhu [9]. In the second step we apply an efficient pseudo-Newton process to go from the approximate path derived in the first step to the true path corresponding to l. In this section we take up the details associated with the first step.

The approximate loss function $\hat{l}$ can be formed by placing *knot points* on the r axis and choosing $\hat{l}''$ to be a constant in each of the intervals defined by the knot points. A look at Figure 1 will help appreciate the approximation ideas that we give below. Since l'' is symmetric about $r = 0$, it is a good idea to choose the knot points to be placed symmetrically about $r = 0$. Thus we can choose positive values, $p_1 < p_2 < \ldots < p_k$ and choose the knot points to be $\{-p_i\}_{i=1}^k \cup \{p_i\}_{i=1}^k$, forming $(2k+1)$ intervals in the r axis. We can also choose $(k+1)$ second derivative values, $\{a_i\}_{i=0}^k$ and define $\hat{l}''$ as: $\hat{l}''(r) = a_0$, if $-p_1 < r < p_1$; and, for $i \geq 1$, $\hat{l}''(r) = a_i$, if $-p_{i+1} < r \leq -p_i$ or $p_i \leq r < p_{i+1}$.[4] Since $\hat{l}$ needs to be convex we require all a_i to be non-negative. Integrating $\hat{l}''$ twice we can get $\hat{l}'$ and $\hat{l}$. This leads to two integration constants which form additional variables. To suit the logistic regression loss function, we enforce the following additional constraints: $a_k = 0$; $\hat{l}'(0) = -0.5$; $\hat{l}'(\infty) = 0$; $\hat{l}'(-\infty) = -1$; and $\hat{l}(\infty) = 0$ (this helps ensure that $\hat{l}$ is a non-negative function). These constraints imply that, for $r \in (-\infty, -p_k]$, $\hat{l}''(r) = 0$, $\hat{l}'(r) = -1$ and, for $r \in [p_k, \infty)$, $\hat{l}''(r) = 0$, $\hat{l}'(r) = 0$. Even after the above mentioned constraints are enforced, there is still freedom left in choosing the $\{p_i\}$ and $\{a_i\}$. Since first derivatives play a very important role in path tracking algorithms, it is a good idea to resolve this freedom by making $\hat{l}'$ as close as possible to l', say, by minimizing the integral of the square of the difference between the two functions. We stress again that this optimization problem is independent of the classification problem being solved. It just needs to be solved once; then the optimized $\hat{l}$ can be used for all problems.

The value of k which defines the approximation also needs to be chosen. Although choosing k to be big will yield excellent approximations, it leads to inefficiencies in path tracking as we will point out later in this section. We have found $k = 2$ to be quite sufficient for all problems that we have tried. The corresponding parameters and the approximations that we have derived are given in Figure 1.

The approximate loss function $\hat{l}$ can be compactly written as $\hat{l}(r) = \frac{1}{2}a(r)r^2 + b(r)r + c(r)$ where $a(r)$, $b(r)$ and $c(r)$ are appropriately defined piecewise constant functions of r (and so their derivatives can be taken to be zero at non-knot points). $a(r)$ is same as $\hat{l}''$ and it plays an important role in the tracking algorithm. $b(r)$ plays a role in gradient calculations. For starting the algorithm we use the fact that $b(0) = -0.5$, which comes from the constraint, $\hat{l}'(0) = -0.5$ that we imposed earlier; as we will see, for the rest of the algorithm, the continuity of gradients allows us to do computations without using the values of $b(r)$ at other values of r. $c(r)$ only leads to an additive constant in the objective function, and so plays no role.

We solve $\min_\beta \hat{f}_\lambda(\beta) = \hat{f}_0(\beta) + \lambda\|\beta\|_1$ where $\hat{f}_0(\beta) = \frac{\mu}{2}\beta^T K\beta + \sum_{i=1}^n \hat{l}(r_i)$. The gradient, $\hat{g}$ and the Hessian, $\hat{H}$ of $\hat{f}_0$ are given by $\hat{g}(\beta) = \mu K\beta + \sum_i [a(r_i)r_i + b(r_i)]t_i x_i$, $\hat{H}(\beta) = \mu K + \sum_i a(r_i)x_i x_i^T$. It is useful to note the following: at $\beta = 0$ we have $r = 0$ and so $\hat{g}(0) = -\sum_i 0.5 t_i x_i$; and, the change, $\delta\hat{g}$ corresponding to a change $\delta\beta$ in β, assuming that no crossing of knot points occur during the change, is given by $\delta\hat{g} = \mu K\delta\beta + \sum_i a(r_i)\delta r_i t_i x_i = \hat{H}\delta\beta$.

[4]For preciseness we take $p_{k+1} = \infty$.

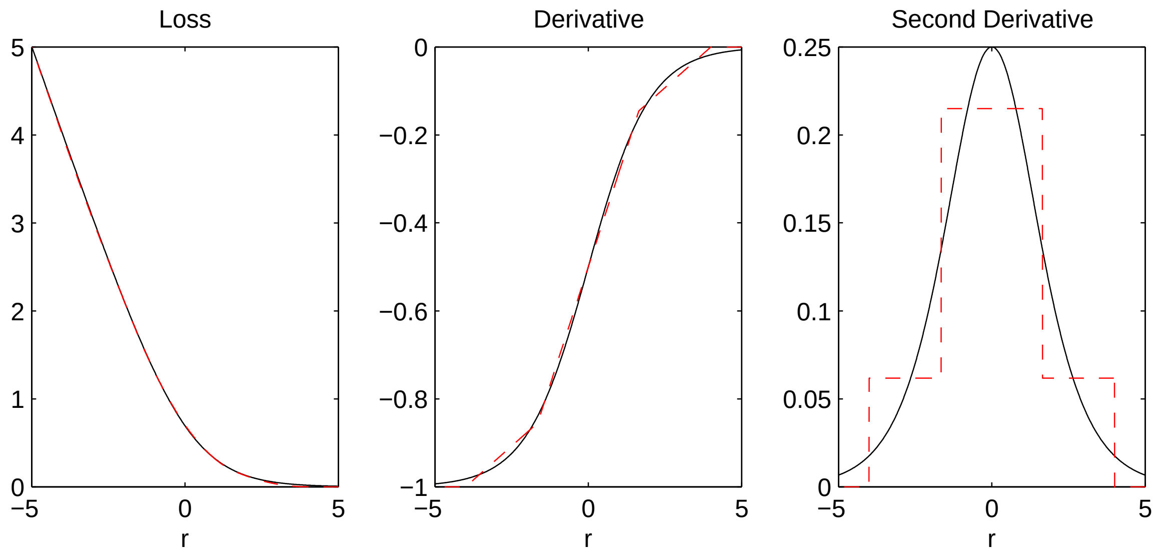

Fig. 1. Approximate loss function defined by the parameters: $k = 2$, $p_1 = 1.65$, $p_2 = 4$, $a_0 = 0.215$, $a_1 = 0.06181$ and $a_2 = 0$. $\hat{l}$, $\hat{l}'$ and $\hat{l}''$ are shown as broken lines while l, l' and l'' are shown as continuous lines.

We now apply the tracking algorithm of Rosset and Zhu [9]. Suppose we are working with some fixed λ and β is the solution. Let $\mathcal{A} = \{j : \beta_j \neq 0\}$ and $\mathcal{A}^c$ be the complement set of $\mathcal{A}$. Let us use the following notations for a vector v: $v_\mathcal{A}$ is the vector of size $|\mathcal{A}|$ containing v_j, $j \in \mathcal{A}$; and, when $v_j \neq 0 \ \forall j$, $\text{sgn}(v)$ is the vector containing the signs of v. For a symmetric matrix X, $X_\mathcal{A}$ will denote the $|\mathcal{A}| \times |\mathcal{A}|$ matrix obtained by keeping only the rows and columns of X corresponding to $\mathcal{A}$. Optimality requires that $\hat{g}_\mathcal{A} + \lambda\, \text{sgn}(\beta_\mathcal{A}) = 0$ and $|\hat{g}_j| \leq \lambda \ \ \forall j \in \mathcal{A}^c$. The first set of equalities define a piecewise linear path for β. Tracking them (typically by decreasing λ) yields a part of the full solution path. This tracking can be done until one of the following occurs: (a) some $j \in \mathcal{A}^c$ gives $|\hat{g}_j| = \lambda$, which means that coefficient corresponding to j has to leave $\mathcal{A}^c$ and enter $\mathcal{A}$; (b) some $j \in \mathcal{A}$ has $\beta_j = 0$, which means that j has to leave $\mathcal{A}$ and go to $\mathcal{A}^c$; and, (c) for some training example index i, r_i hits a knot point, which means that we have to switch $\hat{l}(r_i)$ from one quadratic model to another. The algorithm starts from $\beta = 0$ and repeatedly applies the above ideas to track the entire path. The full algorithm is given below. It is a revised version of *Algorithm 1* of [9] with some changes made to show calculations concerning λ, $\hat{g}$ and r explicitly. It is worth mentioning here that the computational complexity of this algorithm is similar to that of *Algorithm 1* of [9] and is comparable to the training cost associated with finding the solution just at a few selected values of λ [2].

Algorithm for tracking the approximate path.

```
1) Initialize: $\beta = 0$, $r = 0$, $a_i = a(0)\ \forall i$, $\hat{g} = -\sum_i 0.5 t_i x_i$, $\mathcal{A} = \arg\max_j |\hat{g}_j|$, $\lambda = \max_j |\hat{g}_j|$, $\delta\beta_\mathcal{A} = -\hat{H}_\mathcal{A}^{-1}\text{sgn}(\hat{g}_\mathcal{A})$, $\delta\beta_{\mathcal{A}^c} = 0$, $\delta r_i = t_i \delta\beta^T x_i\ \forall i$, $\delta\hat{g} = \hat{H}\,\delta\beta$.
2) While $(\lambda > 0)$
    a) $d_1 = \min\{d > 0 : |\hat{g}_j + d\,\delta\hat{g}_j| = \lambda - d, j \in \mathcal{A}^c\}$
    b) $d_2 = \min\{d > 0 : \beta_j + d\,\delta\beta_j = 0, j \in \mathcal{A}\}$
    c) $d_3 = \min\{d > 0 : r_i + d\,\delta r_i$ hits a knot point for some $i\}$
    d) $d = \min\{d_1, d_2, d_3\}$, $\lambda \leftarrow \lambda - d$, $\beta \leftarrow \beta + d\,\delta\beta$, $r \leftarrow r + d\,\delta r$, $\hat{g} \leftarrow \hat{g} + d\,\delta\hat{g}$
    e) If $d = d_1$ then add coefficient attaining equality at $d$ to $\mathcal{A}$.
    f) If $d = d_2$ then remove coefficient attaining $0$ at $d$ from $\mathcal{A}$.
    g) If $d = d_3$ for example $i^*$, set $a_{i^*}$ to the value of $a(r_{i^*})$ of the new knot zone.
    h) Set $\delta\beta_\mathcal{A} = -\hat{H}_\mathcal{A}^{-1}\text{sgn}(\hat{g}_\mathcal{A})$, $\delta\beta_{\mathcal{A}^c} = 0$, $\delta r_i = t_i\delta\beta^T x_i\ \ \forall i$, $\delta\hat{g} = \hat{H}\,\delta\beta$
```

$\hat{H}$ and its updatings using the a_i are not mentioned in the algorithm, but they should be obvious. Since $\hat{H}_\mathcal{A}^{-1}$ is required in steps (1) and (2h), it is useful to maintain and update a Cholesky decomposition of $\hat{H}_\mathcal{A}$. Changes caused by steps 2(e), 2(f) and 2(g) lead to changes in $\hat{H}_\mathcal{A}$; the corresponding changes to the Cholesky decomposition can be done by standard updating methods. If the number of knot points, $(2k+1)$ is large, then the algorithm will pass through step 2(c) many many times, causing the algorithm to become expensive. The LARS version of the algorithm corresponds to leaving out steps 2(b) and 2(f). In this algorithm, coefficients entering $\mathcal{A}$ never leave it; thus, it is much simpler to implement. Practical experience shows that the LASSO solutions and LARS algorithms produce nearly the same paths. For the rest of the paper we will only consider and use the path obtained from the LARS algorithm. If we assume that the "knot crossing" events occur $O(n)$ times, then every iteration of the above algorithm needs $O(mn)$ effort (for steps 2(a) through 2(d)) and $O(m^2)$ effort for step 2(h). Hence, $O(m+n)$ iterations of the algorithm will require $O(mn^2)$ effort if $m > n$.

III. Pseudo-Newton correction process

The path described in the previous section is already a good approximation of the true path corresponding to logistic regression. But, the quadratic approximation of the loss function is not interpretable as negative log-likelihood. So, it is a good idea to get more closely to the true path, which is done by applying a correction process. Let λ_b and λ_a be the λ values at two consecutive runs of steps 2(a)-2(h). In the interval, (λ_a, λ_b) let $\mathcal{A}$ denote the set of non-zero coefficients. For most applications it is usually sufficient to obtain a precise solution just at the midpoint, $\lambda = (\lambda_a+\lambda_b)/2$.[5] Let us now describe the correction process. At the given λ, we first use the approximate path to get the initial solution, β^0. Let $s_A = \mathrm{sgn}(\hat{g}_{\mathcal{A}}(\beta^0))$. For the LARS version we solve the nonlinear system,

$$g_{\mathcal{A}} = \lambda s_{\mathcal{A}} \tag{5}$$

where $g_{\mathcal{A}}$ is the gradient of f_0 in (3) and is given by

$$g_{\mathcal{A}} = \mu K_{\mathcal{A}} \beta_{\mathcal{A}} - \sum_i \frac{\exp(-r_i)}{1+\exp(-r_i)} t_i x_{\mathcal{A}i} \tag{6}$$

Applying Newton's method on (6) is expensive since that will involve the repeated determination of the Hessian of f_0 and its inverse. Instead, we employ a fixed matrix, $\tilde{H}_{\mathcal{A}}$ that is an approximation of the Hessian, and do the following pseudo-Newton iterations:

$$\beta_{\mathcal{A}}^{t+1} = \beta_{\mathcal{A}}^t - \tilde{H}_{\mathcal{A}}^{-1} g_{\mathcal{A}}, \quad t \geq 0 \tag{7}$$

The cheapest choice for $\tilde{H}_{\mathcal{A}}$ is $\hat{H}_{\mathcal{A}}$, the Hessian of $\hat{f}_0$ which (together with its Cholesky factorization) is available as a by-product of the approximate tracking algorithm of the previous section. Using the tolerance, $\epsilon = 10^{-3}$, we terminate the iterations in (7) when

$$\frac{1}{|\mathcal{A}|} \| g_{\mathcal{A}} - \lambda s_{\mathcal{A}} \| \leq \epsilon \lambda \tag{8}$$

We have also found another construction for $\tilde{H}_{\mathcal{A}}$ that shows much better convergence properties on (7), but requires some extra work. Let us now describe the details of this construction. The exact Hessian of f_0 with respect to $\beta_{\mathcal{A}}$ is given by $H_{\mathcal{A}} = \mu K_{\mathcal{A}} + \sum_i w(r_i) x_{\mathcal{A}i} x_{\mathcal{A}i}^T$ where $w(r_i) = \frac{\exp(-r_i)}{(1+\exp(-r_i))^2}$. While doing the approximate tracking algorithm of section 2, it is usually the case that, in the initial stage when the key coefficients get picked up, the r_i go through a lot of change. But, after the important coefficients get picked up, the r_i (and therefore, the $w(r_i)$) undergo only small changes. With this in mind, we can think of fixing the $w(r_i)$ values at some stage of the algorithm and using it to define $\tilde{H}_{\mathcal{A}}$. For a given choice of $w \in R^n$ let us define

$$\tilde{H}_{\mathcal{A}} = \mu K_{\mathcal{A}} + \sum_i w_i x_{\mathcal{A}i} x_{\mathcal{A}i}^T \tag{9}$$

In the beginning of the approximate tracking algorithm (step 1) we set $w_i = 0.25 \ \forall i$ and compute $\tilde{H}$ (by (9)) and also its factorization (this is just a square root operation since there is only one coefficient). At each pass through step 2(e), $\tilde{H}_{\mathcal{A}}$ is updated to one higher dimension using the w_i. Whenever the correction process is required at some λ, say at the mid-point of an interval, (λ_a, λ_b), we use the current approximation $\tilde{H}_{\mathcal{A}}$ and apply the pseudo-Newton iterations in (7). We allow a maximum of $t_{\max}$ iterations where $t_{\max} = \max\{100, |\mathcal{A}|/100\}$. If the iterations do not converge within those many iterations, that is probably an indication that the w_i that have been used are outdated; so we compute them fresh as $w_i = w(r_i)$, using the current values of r_i, recompute $\tilde{H}_{\mathcal{A}}$ using (9) and also do the Cholesky factorization of $\tilde{H}$ from scratch. Such a process works quite efficiently. Non-convergence of (7) within $t_{\max}$ iterations (which prompts the complete re-determination of $\tilde{H}$ and its factorization) takes place only occasionally and so the whole process is efficient.

For the LASSO version, we also need to watch if, during the application of (7), a LASSO optimality condition is violated, e.g., a β_i^t, $i \in \mathcal{A}$ changes sign. Although such occurrences are rare, suitable additional steps need to be added to ensure LASSO optimality. Our future implementation will include these steps.

IV. Numerical experiments

To give an idea of the speed of our tracking method, we compare it against the BBR software of Genkin et al [4]. (In the future we intend to do a comparison also with the GenLASSO code of [11].) BBR employs a cyclic coordinate descent method and has been shown to be robust and efficient on large text classification problems. Because BBR does not allow a mix of L_1 and L_2 regularizers, we do all the experiments for the case, $\mu = 0$. We use four datasets. The first two, *Fbis-4* ($n = 2463$, $m = 2000$) and *Ohscal-1* ($n = 11162$, $m = 11465$) are taken from the text categorization domain [5] where the number of features is very large. For these datasets linear classifiers work very well and so we employ the sparse logistic regression models directly on the input variables. The next two datasets, *Waveform* and *Splice* [8], require nonlinear classification and so we employ KLR, with the kernel function given by $k(z_i, z_j) = \exp(-\gamma \|z_i - z_j\|^2)$. For *Waveform*, $n = 400$, dimension of z is 21, and $\gamma = 0.0625$. For *Splice*, $n = 1000$, dimension of z is 7,[6] and $\gamma = 0.2$. (In each case, the chosen γ value corresponds roughly to the value at which the generalization error is best.) For each dataset we chose a value of λ at which the 10-fold cross validation error rate is quite good and compared the CPU times required by the two methods to get the solution at this λ.

Because BBR and our method use different stopping criteria, we proceed as follows for doing the comparison. BBR terminates when the relative change in the r values over one cycle of coordinate descent steps is smaller than a tolerance, *eps*. Since the BBR software allows the specification of a value of *eps*, we varied *eps* over a range of values and chose that value at which the solution of BBR satisfies our criterion in (8). The CPU time needed by BBR for this *eps*

[5] If precise solutions are needed at other values of λ in that interval, then the correction process described below can be repeated at those values. If a full path is needed, we can first get the precise solutions at a few points in the interval using the correction process and then apply an interpolation technique.

[6] Although the number of original input variables of this dataset is 60, we only used variables 28-34 since they are the only relevant ones.

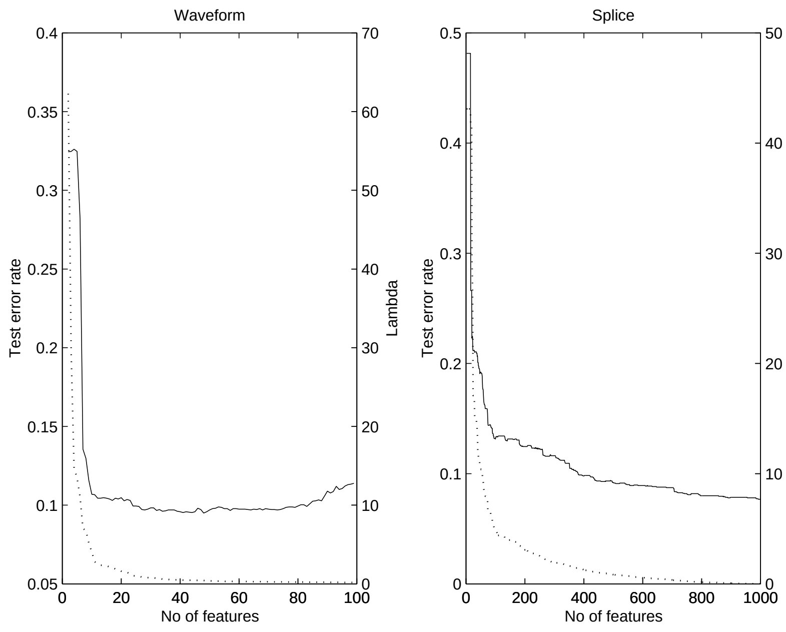

Fig. 2. Variation of test error rate (continuous line) and λ (dotted line) as a function of the number of features (basis functions) chosen, for the *Waveform* and *Splice* datasets.

TABLE I

COMPARISON OF THE EFFICIENCY OF OUR METHOD AGAINST BBR. T_APPROX AND T_CORR ARE THE TIMES REQUIRED BY OUR METHOD FOR THE APPROXIMATE TRACKING AND THE CORRECTION PROCESS; T_TOTAL=T_APPROX+T_CORR. T_BBR IS THE TIME NEEDED BY BBR. ALL TIMES ARE IN SECONDS ON A 2.5 GHZ MACHINE.

		Our Method			**BBR**
Dataset	λ	T_Approx	T_Corr	T_Total	T_BBR
Fbis-4	3.5	20.0	5.5	25.5	131
Ohscal-1	4.0	141.1	18.1	159.2	88
Waveform	5.0	2.9	0.7	3.6	4642
Splice	1.0	39.3	21.4	60.7	78685

was compared to the CPU time needed by our method. Table 1 gives the results. Clearly, our method is quite efficient. BBR is efficient for text classification, but it seems to be inefficient for KLR, possibly due to the ill-conditioning of the Hessian caused by the closeness of the training examples. It should be noted that, while BBR was run to obtain the solution at the specified λ only, our method tracks the solution till that λ; more specifically, as mentioned at the beginning of section 3, it gets the solution at the midpoint of each interval,[7] (λ_a, λ_b) mentioned there, until the specified λ value is reached.

Figure 2 gives plots of λ and test error rate as functions of the number of examples chosen, for *Waveform* (3600 test examples) and *Splice* (2175 test examples). For the *Waveform* dataset, a small fraction of the kernel basis functions (about one-tenth) seems to be sufficient for obtaining the best generalization, while the *Splice* dataset requires a large fraction of the basis functions. The sparse logistic regression approach (combined with cross validation) can be used to infer these effectively. It is useful to note from the plots for *Waveform* in Figure 2 that, at the point where test error rate stabilizes nearly to a flat curve, λ is still varying sharply. So it is difficult to determine, beforehand, the λ value at which the error rate stabilizes. Methods such as BBR will have to try several values of λ in order to stop at the right point, whereas, in our tracking method, determining this point is easily done.

[7]For each of the datasets there are hundreds of such intervals.

V. CONCLUSION

In this paper we proposed an algorithm for tracking the solution curve of sparse logistic regression. The algorithm is based on approximating the logistic regression loss by a piecewise quadratic function, tracking the piecewise linear solution curve corresponding to it, and then applying a correction step to get to the true path. Experimental results on benchmark datasets suggest that the proposed algorithm is fast and so it is a useful tool for doing feature selection in linear classifiers and for designing nonlinear classifiers via sparse kernel logistic regression.

REFERENCES

[1] Bach, F.R., Thibaux, R. & Jordan, M. I. (2005) Computing regularization paths for learning multiple kernels. In NIPS-17, Cambridge, MA: MIT Press.

[2] Efron, B., Hastie, T., Johnstone, T. & Tibshirani, R. (2004) Least angle regression. *Annals of Statistics*, **32**(2):407-451.

[3] Friedman, J.H. & Popescu, B.E. (2004) Gradient directed regularization. Technical Report, Department of Statistics, Stanford University, Stanford.

[4] Genkin, A., Lewis, D.D. & Madigan, D. (2004) Large-scale Bayesian logistic regression for text categorization. Technical Report, DIMACS, New Jersey.

[5] Keerthi, S.S. (2005) Generalized LARS as an effective feature selection tool for text classifi cation with SVMs. In *Proceedings of the 22nd International Conference on Machine Learning.*

[6] Madigan, D. & Ridgeway, G. (2004) Discussion of "Least angle regression" by Efron, Johnstone, Hastie, and Tibshirani. *Annals of Statistics* **32**(2):465-468.

[7] Osborne, M., Presnell, B. & Turlach, B. (2000) On the Lasso and its dual. *Journal of Computational and Graphical Statistics* **9**(2):319-337.

[8] Rätsch, G. (1998-2005). Benchmark repository. *http://ida.fi rst.fraunhofer.de/∼raetsch/*

[9] Rosset, S. & Zhu, J. (2004) Piecewise linear regularized solution paths. Technical Report, Stanford University, Stanford.

[10] Rosset, S. (2005) Tracking curved regularized optimization solution paths. In *Advances in Neural Information Processing Systems 17*. Cambridge, MA: MIT Press.

[11] Roth, V. (2002) The Generalized LASSO: a wrapper approach to gene selection for microarray data. IEEE Transactions on Neural Networks, Vol. 15, 2004.

[12] Shevade, S.K. & Keerthi, S.S. (2003) A simple and effi cient algorithm for gene selection using sparse logistic regression. *Bioinformatics* **19**(17):2246-2253.

[13] Tibshirani, R. (1996) Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, B* **58**:267-288.

[14] Zhu, J. & Hastie, T. (2005) Kernel logistic regression and the import vector machine. *Journal of Computational and Graphical Statistics* **14**:185-205.

[15] Zou, H. & Hastie, T. (2005) Regularization and variable selection via the Elastic Net. *JRSSB*, Vol. 67, pp.301-320.