The Kernel Path in Kernelized LASSO*

Gang Wang, Dit-Yan Yeung, Frederick H. Lochovsky
Department of Computer Science and Engineering
Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong, China

Abstract

Kernel methods implicitly map data points from the input space to some feature space where even relatively simple algorithms such as linear methods can deliver very impressive performance. Of crucial importance though is the choice of the kernel function, which determines the mapping between the input space and the feature space. The past few years have seen many efforts in learning either the kernel function or the kernel matrix. In this paper, we study the problem of learning the kernel hyperparameter in the context of the kernelized LASSO regression model. Specifically, we propose a solution path algorithm with respect to the hyperparameter of the kernel function. As the kernel hyperparameter changes its value, the solution path can be traced exactly without having to train the model multiple times. As a result, the optimal solution can be identified efficiently. Some simulation results will be presented to demonstrate the effectiveness of our proposed kernel path algorithm.

1 Introduction

Kernel methods (Schölkopf & Smola, 2002; Müller et al., 2001) have demonstrated great successes in solving many machine learning and pattern recognition problems. These methods implicitly map data points from the input space to some feature space where typically a linear method is applied. The implicit feature mapping is determined by a kernel function, which allows the inner product between two points in the feature space to be computed without having to know the explicit mapping from the input space to the feature space. Rather, it is simply a function of two points in the input space.

For a kernel method to perform well, the kernel function often plays a very crucial role. Rather than choosing the kernel function and setting its hyperparameters manually, many attempts have been made over the past few years to automate this process, at least partially. These methods can generally be categorized into two major approaches based on either kernel function learning or kernel matrix learning. The kernel function learning approach ranges from learning the hyperparameters of some prespecified kernel function to learning a kernel function within a class of possible kernel functions. The method of hyperkernels (Ong et al., 2005) belongs to the latter extreme which is more general than the former extreme. However, the number of hyperparameters that need to be learned is so large that it is not very practical for many real-world applications. Instead of learning the kernel function itself, the second approach bypasses it by learning the kernel matrix defined only for a given set of data points. A number of kernel matrix learning methods have been proposed, e.g., (Cristianini et al., 2002; Lanckriet et al., 2004; Zhang et al., 2006). However, many of these methods are based on the transductive learning setting, making it nontrivial to achieve out-of-sample extension. Moreover, many of these methods are formulated as semidefinite programming (SDP) problems. Despite their convexity, they incur high computational cost especially when the data set is large.

Our paper adopts the kernel function learning approach. However, unlike the method of hyperkernels, we seek to learn the optimal hyperparameter value for a prespecified kernel function. The traditional approach to this model selection problem is to apply methods like cross validation to determine the best choice among a number of prespecified hyperparameter values. Extensive exploration such as performing

*This research is supported by Competitive Earmarked Research Grant (CERG) 621706 from the Research Grants Council (RGC) of the Hong Kong Special Administrative Region, China.

line search for one hyperparameter or grid search for two hyperparameters is usually used. However, this requires training the model multiple times with different hyperparameter values and hence is computationally prohibitive especially when the number of candidate values is very large.

More recently, a novel approach has emerged that seeks to explore the entire solution path for all regularization parameter values without having to train the model multiple times. Efron et al. (2004) developed the least angle regression (LARS) algorithm which fits the coefficient path for the linear least square regression problem regularized with the L_1 norm. An important finding is that the coefficient path is piecewise linear and hence it is efficient to explore the entire solution path by monitoring the breakpoints only. Zhu et al. (2003) proposed an algorithm to compute the entire regularization path for the L_1-norm support vector classification (SVC) and Hastie et al. (2004) proposed one for the standard L_2-norm SVC. They are again based on the property that the regularization paths are piecewise linear. More generally, Rosset and Zhu (2003) showed that any model with an L_1 regularization and a quadratic, piecewise quadratic, piecewise linear, or linear loss function has a piecewise linear coefficient path and hence the entire solution path can be computed efficiently. For general loss functions and regularization constraints, the solution paths are typically not piecewise linear. Bach et al. (2005) explored a nonlinear regularization path for multiple kernel learning regularized with a block L_1-norm. Rosset (2004) proposed a general path following algorithm to approximate the original regularization path for such cases. Besides for the regularization parameter, our recent work (Wang et al., 2006) shows that this approach can also be used for exploring the solution path for some other hyperparameter.

In this paper, we propose a novel method that traces the entire solution path for the kernel hyperparameter of a kernel-based nonlinear regression method, called kernelized LASSO (kLASSO) (Roth, 2004). Since the kernel hyperparameter is embedded into each entry of the kernel matrix, the next breakpoint cannot be computed beforehand. Moreover, the path is no longer piecewise linear. Nevertheless, our method provides an efficient algorithm that can still calculate the next optimal solution exactly as the kernel hyperparameter value changes. Hence, we can trace the kernel path exactly instead of using the approximation technique of (Rosset, 2004). Moreover, the algorithm is a general approach that can be used for many different kernel functions.

The rest of this paper is organized as follows. In Section 2, we review the kLASSO model and briefly discuss the regularization path algorithm. In Section 3, we derive the updating equation for the optimal solution with respect to the kernel hyperparameter and present the kernel path algorithm. More discussions of the kernel path algorithm are given in Section 4 and some experimental results are presented in Section 5. Finally, Section 6 concludes the paper.

2 Kernelized LASSO

In a typical regression problem, we are given a training set of independent and identically distributed (iid) data pairs $\{(\mathbf{x}_i, y_i)\}_{i=1}^n \subset \mathbb{R}^d \times \mathbb{R}$, where $\mathbf{x}_i$ and y_i are the input and output, respectively, of the ith pair. A special form of regression called LASSO regression was proposed by Tibshirani (1996), where the acronym LASSO stands for *least absolute shrinkage and selection operator*. LASSO regularizes ordinary least square regression with an L_1 regularizer. It is an effective linear regression technique for feature selection. It can lead to sparse solutions by shrinking the coefficients of the irrelevant or redundant features to zero. Theoretical analysis (Ng, 2004) shows that LASSO is particularly effective when there exist many irrelevant features but only very few training examples.

2.1 Model Formulation

The kernelized LASSO (kLASSO) (Roth, 2004) uses $\mathcal{D} = \{K(\mathbf{x}, \mathbf{x}_1), ..., K(\mathbf{x}, \mathbf{x}_n)\}$ as a dictionary of basis functions to extend LASSO for nonlinear regression, where $K_\gamma(\mathbf{x}_i, \mathbf{x}_j)$ is the kernel function defined on $\mathbb{R}^d \times \mathbb{R}^d$ and γ is some kernel hyperparameter in the kernel function. Thus, a representation of the regression function f admits the following form:

$$f(\mathbf{x}) = \sum_{i=1}^n \beta_i K_\gamma(\mathbf{x}_i, \mathbf{x}) + \beta_0. \qquad (1)$$

where $\boldsymbol{\beta} = (\beta_i)_{i=1}^n$ and β_0 are the coefficients of the regression function. Let $\mathbf{y} = (y_i)_{i=1}^n$ denote the output vector and $\mathbf{K} = [K(\mathbf{x}_i, \mathbf{x}_j)]_{i,j=1}^n$ the kernel matrix. The regularized empirical loss L_k that kLASSO aims to minimize is given by

$$L_k(\boldsymbol{\beta}, \beta_0) = \frac{1}{2}\|\mathbf{y} - \mathbf{K}\boldsymbol{\beta} - \beta_0\mathbf{1}\|_2^2 + \lambda\|\boldsymbol{\beta}\|_1, \qquad (2)$$

where $\mathbf{1}$ is a vector of ones, $\|\cdot\|_2$ denotes the L_2 norm of a vector, $\|\cdot\|_1$ denotes the L_1 norm of a vector, and λ is a regularization parameter. Note that $f(\mathbf{x})$ is expressed as an expansion in terms of only a subset of the data points for which β_i is nonzero. Since L_1 norm is used, $f(\mathbf{x})$ tends to give a sparse representation. There are two hyperparameters, i.e., γ and λ, in kLASSO.

Taking the first derivative of L_k in (2) with respect to β_0 and setting it to zero, we get

$$\beta_0 = (\mathbf{y} - \mathbf{K}\boldsymbol{\beta})^T \mathbf{1}/n. \tag{3}$$

Hence the optimal β_0 can be calculated directly from $\boldsymbol{\beta}$. On the other hand, L_k is not differentiable with respect to $\boldsymbol{\beta}$ since the L_1 norm is used. This calls for a special method to overcome the problem. Following Rosset and Zhu (2003), we represent $\boldsymbol{\beta} = \boldsymbol{\beta}^+ - \boldsymbol{\beta}^-$, where $\boldsymbol{\beta}^+ = (\beta_i^+)_{i=1}^n$ and $\boldsymbol{\beta}^- = (\beta_i^-)_{i=1}^n$ with the constraints $\beta_i^+, \beta_i^- \geq 0$ for $i = 1, \ldots, n$. The regularized optimization problem can thus be rewritten as

$$\begin{aligned} \min_{\boldsymbol{\beta}^+, \boldsymbol{\beta}^-} \quad & \frac{1}{2} \left\| \mathbf{y} - \mathbf{K}(\boldsymbol{\beta}^+ - \boldsymbol{\beta}^-) - \beta_0 \mathbf{1} \right\|_2^2 \\ & + \lambda \sum_{i=1}^n (\beta_i^+ + \beta_i^-) \qquad (4) \\ \text{subject to} \quad & \beta_i^+, \beta_i^- \geq 0 \quad i = 1, \ldots, n. \qquad (5) \end{aligned}$$

Introducing Lagrange multipliers $\boldsymbol{\mu}^+ = (\mu_i^+)$ and $\boldsymbol{\mu}^- = (\mu_i^-)$ $(i = 1, \ldots, n)$ for the inequality constraints, the Lagrangian dual function can be expressed as

$$\begin{aligned} L_d \; = \; & \frac{1}{2} \left\| \mathbf{y} - \mathbf{K}(\boldsymbol{\beta}^+ - \boldsymbol{\beta}^-) - \beta_0 \mathbf{1} \right\|_2^2 \qquad (6) \\ & + \lambda \sum_{i=1}^n (\beta_i^+ + \beta_i^-) - \sum_{i=1}^n \mu_i^+ \beta_i^+ - \sum_{i=1}^n \mu_i^- \beta_i^- . \quad (7) \end{aligned}$$

Since L_d is differentiable with respect to both $\boldsymbol{\beta}^+$ and $\boldsymbol{\beta}^-$, we can state the optimality conditions as:

$$\frac{\partial L_d}{\partial \beta_i^+} = \big(-\mathbf{K}(\mathbf{y} - \mathbf{K}(\boldsymbol{\beta}^+ - \boldsymbol{\beta}^-) - \beta_0 \mathbf{1}) \big)_i + \lambda - \mu_i^+ = 0 \tag{8}$$

$$\frac{\partial L_d}{\partial \beta_i^-} = \big(\mathbf{K}(\mathbf{y} - \mathbf{K}(\boldsymbol{\beta}^+ - \boldsymbol{\beta}^-) - \beta_0 \mathbf{1}) \big)_i + \lambda - \mu_i^- = 0. \tag{9}$$

From the KKT conditions, we have

$$\mu_i^+ \beta_i^+ = 0, \quad \mu_i^+ \geq 0, \tag{10}$$

$$\mu_i^- \beta_i^- = 0, \quad \mu_i^- \geq 0. \tag{11}$$

Based on the conditions (8)–(11) above, we know that, for any fixed hyperparameter values, the optimal solution should have the following properties:

- If $\lambda > 0$:
  - $\hat{\beta}_i^+ > 0 \Rightarrow \Big(\mathbf{K}(\mathbf{y} - \mathbf{K}(\hat{\boldsymbol{\beta}}^+ - \hat{\boldsymbol{\beta}}^-) - \hat{\beta}_0 \mathbf{1}) \Big)_i = \lambda, \; \beta_i^- = 0$
  - $\hat{\beta}_i^- > 0 \Rightarrow \Big(\mathbf{K}(\mathbf{y} - \mathbf{K}(\hat{\boldsymbol{\beta}}^+ - \hat{\boldsymbol{\beta}}^-) - \hat{\beta}_0 \mathbf{1}) \Big)_i = -\lambda, \; \hat{\beta}_i^+ = 0$
  - $\hat{\beta}_i^+ = 0, \; \hat{\beta}_i^- = 0 \Rightarrow -\lambda \leq \Big(\mathbf{K}(\mathbf{y} - \mathbf{K}(\hat{\boldsymbol{\beta}}^+ - \hat{\boldsymbol{\beta}}^-) - \hat{\beta}_0 \mathbf{1}) \Big)_i \leq \lambda$
- If $\lambda = 0$:
  - $\mathbf{K}(\mathbf{y} - \mathbf{K}\hat{\boldsymbol{\beta}} - \hat{\beta}_0 \mathbf{1}) = \mathbf{0}$ (non-regularized solution)

Let $\mathbf{g} = (g_i)_{i=1}^n$ be an n-dimensional vector such that $\mathbf{g} = \mathbf{K}(\mathbf{y} - \mathbf{K}\boldsymbol{\beta} - \beta_0 \mathbf{1})$. For those nonzero coefficients $\hat{\beta}_i$ at the optimal solution, their corresponding elements g_i are equal to λ or $-\lambda$, whose sign is determined by the sign of the coefficient. Thus, we have a set of active points, denoted as $\mathcal{A} = \{ i \in \{1, ..., n\} : \hat{\beta}_i \neq 0 \}$, such that

$$i \in \mathcal{A} \;\; \Rightarrow \;\; g_i = \operatorname{sgn}(\hat{\beta}_i)\lambda \tag{12}$$

$$i \notin \mathcal{A} \;\; \Rightarrow \;\; |g_i| < \lambda. \tag{13}$$

As the value of a hyperparameter changes, the direction along which $\hat{\beta}$ moves should also satisfy conditions (12) and (13). Based on these properties, we can derive either the regularization path algorithm or the kernel path algorithm for kLASSO.

2.2 Regularization Path Algorithm

We first briefly review the regularization path algorithm for kLASSO. The initialization of the λ-path is quite straightforward. When λ tends to infinity initially, the optimal solution corresponds to $\boldsymbol{\beta} = \mathbf{0}$ and $\beta_0 = \mathbf{y}^T \mathbf{1}/n$. Thus $\mathbf{g} = \mathbf{K}(\mathbf{y} - \beta_0 \mathbf{1})$. At this moment, the active set $\mathcal{A}$ is empty and hence condition (13) holds for all elements in $\mathbf{g}$. More accurately, we have $|g_i| < \lambda \; \forall i$. Thus, the regression function value is β_0 for any input and hence it is flat. As λ decreases, all elements in $\boldsymbol{\beta}$ will remain to be zero until λ reaches a certain value when one data point has its $|g_i|$ value equal to λ. This data point is then added to $\mathcal{A}$.

As λ further decreases, the coefficient $\boldsymbol{\beta}_{\mathcal{A}} = (\beta_i) \; \forall i \in \mathcal{A}$ will coordinate accordingly to ensure that $|\mathbf{g}_i| = \lambda \; \forall i \in \mathcal{A}$. Generally, the updating direction of the optimal solution $\frac{\partial \hat{\boldsymbol{\beta}}}{\partial \lambda}$ can be calculated directly from condition (12) when $\mathcal{A}$ is fixed. However, as λ decreases, some data points may join $\mathcal{A}$ while some other data points in $\mathcal{A}$ may leave it. When $\mathcal{A}$ changes, the updating direction has to be re-calculated. The regularization algorithm monitors the breakpoint at which $\mathcal{A}$ changes and calculates a new updating direction for the next solution. For a kLASSO solution with a certain λ value, the next λ breakpoint can be computed before the regularization path extends to this point. Since the path proceeds linearly along the given direction, any solution between two breakpoints can be computed directly.

3 Kernel Path Algorithm

Let γ be the hyperparameter in the kernel function $K_\gamma(\mathbf{x}_i, \mathbf{x}_j)$. The kernel matrix $\mathbf{K}^\gamma$ varies as γ changes. The principle underlying the update of solution in the kernel path is the same as that for the regularization path, i.e., the updating direction of the optimal solution $\frac{\partial \hat{\boldsymbol{\beta}}}{\partial \gamma}$ should also satisfy conditions (12) and (13). Let us consider the period between the lth event (with $\gamma = \gamma^l$) and the $(l+1)$th event (with $\gamma = \gamma^{l+1}$). The active set $\mathcal{A}$ remains unchanged during this period with no point joining or leaving the set. Suppose $\mathcal{A}^l$ contains m indices which can be represented as an m-tuple $(\mathcal{A}^l(1), ..., \mathcal{A}^l(m))$ such that $\mathcal{A}^l(i) < \mathcal{A}^l(j)$ for $i < j$. Let $(\boldsymbol{\beta}^l, \beta_0^l)$ and $\mathcal{A}^l$ denote the optimal solution and the active set, respectively, right after the lth event has occurred. For any γ value with $\gamma^{l+1} < \gamma < \gamma^l$, the new solution $(\boldsymbol{\beta}, \beta_0)$ should satisfy

$$\begin{aligned} -\beta_0 + (\mathbf{y} - \mathbf{K}^\gamma \boldsymbol{\beta})^T \mathbf{1}/n &= 0 && (14) \\ (\mathbf{K}^\gamma(\mathbf{y} - \mathbf{K}^\gamma \boldsymbol{\beta} - \beta_0 \mathbf{1}))_{\mathcal{A}^l} &= \operatorname{sgn}(\boldsymbol{\beta}^l_{\mathcal{A}^l})\lambda. && (15) \end{aligned}$$

Let $\mathbf{k}_i$ be the ith column of the kernel matrix $\mathbf{K}$. Then $\mathbf{K}_{\mathcal{A}^l} = [\mathbf{k}_{\mathcal{A}^l(1)}, ..., \mathbf{k}_{\mathcal{A}^l(m)}]$ is an $n \times m$ matrix and $\mathbf{H}_{\mathcal{A}^l} = \mathbf{K}^T_{\mathcal{A}^l}\mathbf{K}_{\mathcal{A}^l}$ is an $m \times m$ matrix. Since the coefficients for points outside $\mathcal{A}$ are zero, equations (14) and (15) can be simplified to

$$\begin{aligned} \beta_0 + (\mathbf{K}^\gamma_{\mathcal{A}^l}\boldsymbol{\beta}_{\mathcal{A}^l})^T\mathbf{1}/n &= \mathbf{y}^T\mathbf{1}/n && (16) \\ \beta_0(\mathbf{K}^\gamma_{\mathcal{A}^l})^T\mathbf{1} + \mathbf{H}^\gamma_{\mathcal{A}^l}\boldsymbol{\beta}_{\mathcal{A}^l} &= (\mathbf{K}^\gamma_{\mathcal{A}^l})^T\mathbf{y} - \operatorname{sgn}(\boldsymbol{\beta}^l_{\mathcal{A}^l})\lambda (17) \end{aligned}$$

As the objective is to obtain a new solution $(\boldsymbol{\beta}, \beta_0)$ from $(\boldsymbol{\beta}^l, \beta_0^l)$, we transform equation (16) to

$$\begin{aligned} (\beta_0 - \beta_0^l) + (\mathbf{K}^\gamma_{\mathcal{A}^l}(\boldsymbol{\beta}_{\mathcal{A}^l} - \boldsymbol{\beta}^l_{\mathcal{A}^l}))^T\mathbf{1}/n &= \\ -\beta_0^l - (\mathbf{K}^\gamma_{\mathcal{A}^l}\boldsymbol{\beta}^l_{\mathcal{A}^l})^T\mathbf{1}/n + \mathbf{y}^T\mathbf{1}/n & \qquad (18) \end{aligned}$$

and equation (17) to

$$\begin{aligned} (\beta_0 - \beta_0^l)(\mathbf{K}^\gamma_{\mathcal{A}^l})^T\mathbf{1} + \mathbf{H}^\gamma_{\mathcal{A}^l}(\boldsymbol{\beta}_{\mathcal{A}^l} - \boldsymbol{\beta}^l_{\mathcal{A}^l}) &= \\ -\beta_0^l(\mathbf{K}^\gamma_{\mathcal{A}^l})^T\mathbf{1} - \mathbf{H}^\gamma_{\mathcal{A}^l}\boldsymbol{\beta}^l_{\mathcal{A}} + (\mathbf{K}^\gamma_{\mathcal{A}^l})^T\mathbf{y} - \operatorname{sgn}(\boldsymbol{\beta}^l_{\mathcal{A}^l})\lambda.&(19) \end{aligned}$$

Hence, (18) and (19) constitute $m+1$ linear equations. We define the following variables:

$$\Delta^a = \begin{bmatrix} \beta_0 - \beta_0^l \\ \boldsymbol{\beta}_{\mathcal{A}^l} - \boldsymbol{\beta}^l_{\mathcal{A}^l} \end{bmatrix}, \ \mathbf{A} = \begin{bmatrix} 1 & \mathbf{1}^T\mathbf{K}^\gamma_{\mathcal{A}^l}/n \\ (\mathbf{K}^\gamma_{\mathcal{A}^l})^T\mathbf{1} & \mathbf{H}^\gamma_{\mathcal{A}^l} \end{bmatrix},$$

$$\mathbf{b}^a = \begin{bmatrix} -\beta_0^l - (\mathbf{K}^\gamma_{\mathcal{A}^l}\boldsymbol{\beta}^l_{\mathcal{A}^l})^T\mathbf{1}/n + \mathbf{y}^T\mathbf{1}/n \\ -\beta_0^l(\mathbf{K}^\gamma_{\mathcal{A}^l})^T\mathbf{1} - \mathbf{H}^\gamma_{\mathcal{A}^l}\boldsymbol{\beta}^l_{\mathcal{A}} + (\mathbf{K}^\gamma_{\mathcal{A}^l})^T\mathbf{y} - \operatorname{sgn}(\boldsymbol{\beta}^l_{\mathcal{A}^l})\lambda \end{bmatrix}.$$

Then the $m+1$ linear equations above can be represented in matrix form as

$$\mathbf{A}\Delta^a = \mathbf{b}^a \qquad (20)$$

If $\mathbf{A}$ is of full rank, then $\mathbf{A}^{-1}$ exists and we have

$$\begin{bmatrix} \beta_0 \\ \boldsymbol{\beta}_{\mathcal{A}^l} \end{bmatrix} = \begin{bmatrix} \beta_0^l \\ \boldsymbol{\beta}^l_{\mathcal{A}^l} \end{bmatrix} + \begin{bmatrix} c_0 \\ \mathbf{c} \end{bmatrix}, \qquad (21)$$

where $\begin{bmatrix} c_0 \\ \mathbf{c} \end{bmatrix} = \mathbf{c}^a = \mathbf{A}^{-1}\mathbf{b}^a$. As a result, equation (21) gives the updating formula for the new solution. Unlike most solution path algorithms in which the path is piecewise linear, the solution $(\boldsymbol{\beta}, \beta_0)$ in the kernel path does not proceed linearly in γ. Despite this seemingly undesirable property, it is interesting to note that the new solution can still be calculated exactly. The updating direction $\mathbf{c}^a$ depends only on the new kernel value γ and the old solution $(\boldsymbol{\beta}^l, \beta_0)$, but does not depend on the old kernel value γ^l.

From equation (21), we can update the coefficients of the points in $\mathcal{A}$ while other coefficients remain zero. As γ changes, the algorithm needs to monitor the occurrence of any of the following events:

- One of the $\beta_{\mathcal{A}(i)}$ for $i = 1, \dots, m$ reaches 0;
- A point $i \notin \mathcal{A}$ joins $\mathcal{A}$, i.e., $|g_i| = \lambda$.

By monitoring the occurrence of these events, we compute the largest $\gamma < \gamma^l$ for which an event occurs. This γ value is a breakpoint and is denoted as γ^{l+1}. We then update $\mathcal{A}$ and continue until the algorithm terminates.

For those solution path algorithms in which the path is piecewise linear with respect to a hyperparameter, the breakpoint at which the next event occurs can be calculated in advance before actually reaching it. However, the value of the kernel hyperparameter is implicitly embedded into the pairwise distance between points. As a result, we need to specify a γ value in advance to compute the next solution and then check whether the next event has occurred or not. Suppose we are given the optimal solution when $\gamma = \gamma^l$. We propose here an efficient algorithm for estimating the next breakpoint, i.e., γ^{l+1}, at which the next event will occur. Table 1 shows the pseudocode for the kernel path algorithm. The user has to provide a decay rate $\theta \in (0, 1)$ in advance. At each iteration, γ is decreased through multiplying it by θ. If the next event has not occurred, we continue to multiply γ by θ. Otherwise the decay rate is set to $\theta^{1/2}$. The above steps are repeated until the decay rate becomes less than $(1 - \epsilon)$, where ϵ is some error tolerance specified by the user in advance. Hence, we can estimate the breakpoint γ such that $\gamma^{l+1}(1 - \epsilon) \leq \gamma \leq \gamma^{l+1}$. Note that this algorithm only describes the kernel path algorithm in the decreasing direction. In general, γ can either increase or decrease. The kernel path algorithm for the increasing direction is very similar and hence omitted here due to space limitation. Figure 1 shows part of the coefficient paths during the execution of the kernel path algorithm when the regularization parameter is fixed.

Table 1: Kernel path algorithm for the decreasing direction.

```
Input: $\hat{\boldsymbol{\beta}}$, $\hat{\beta}_0$, $\gamma^0$ (initial solution for $\gamma$)
       $\theta$, $\epsilon$, $\gamma_{min}$ (decay rate, error tolerance,
       and $\gamma$ limit)

1. set $t = 0$; set $\boldsymbol{\beta}^0 = \hat{\boldsymbol{\beta}}$, $\beta_0^0 = \hat{\beta}_0$;
2. while ($\gamma > \gamma_{min}$)
3.    set $r = \theta$;
4.    while ($r < 1 - \epsilon$)
5.      $\gamma = \gamma_t r$;
6.      use (21) to compute $(\boldsymbol{\beta}(\gamma), \beta_0(\gamma))$;
7.      if $(\boldsymbol{\beta}(\gamma), \beta_0(\gamma))$ is the valid solution
8.        $\boldsymbol{\beta}^{t+1} = \boldsymbol{\beta}(\gamma)$; $\beta_0^{t+1} = \beta_0(\gamma)$;
9.        $\gamma_{t+1} = \gamma$; $t = t + 1$;
10.     else $r = r^{1/2}$;
11.     endif
12.   end-while
13.   update $\mathcal{A}$;
14. end-while;

Output: a sequence of solutions $(\boldsymbol{\beta}(\gamma), \beta_0(\gamma))$
with $\gamma_{min} < \gamma < \gamma_0$
```

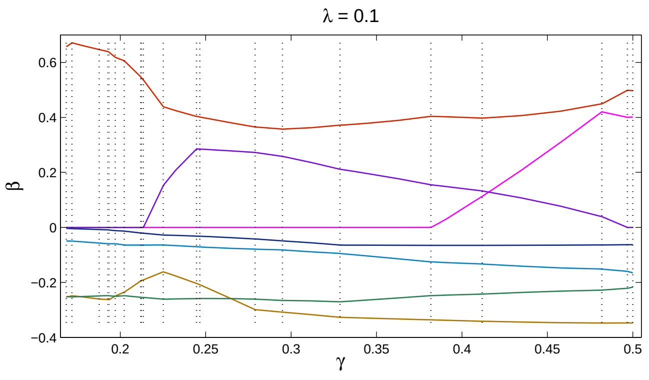


Figure 1: An example of the coefficient paths in kLASSO. The RBF kernel hyperparameter increases from 0.16 to 0.5 when λ is fixed at 0.1. The vertical lines indicate the breakpoints in the coefficient $\boldsymbol{\beta}(\gamma)$ paths.

We assume on average that the ratio of the γ values (γ^{t+1}/γ^t) at two consecutive events is π. Thus, the algorithm needs $(\log_\theta \pi + \log_2(\log_{1-\epsilon}\theta))$ iterations from γ^t to γ^{t+1}. The choice of θ is a tradeoff between $\log_\theta \pi$ and $\log_2(\log_{1-\epsilon}\theta)$, and the choice of ϵ represents a tradeoff between computational complexity and accuracy. Figure 2 shows the number of iterations needed as a function of the decay rate θ. Three average ratios, $\pi = 0.85, 0.9$ and 0.95, are considered and the error tolerance ϵ is set to 10^{-6}. If θ is chosen to be not very close to 1, then the number of iterations does not change much. Thus, we set $\theta = 0.95$ and $\epsilon = 10^{-6}$ in our experiment, and the algorithm always takes less than 20 iterations to reach the next breakpoint. In the 7th step, the algorithm checks whether the new solution is valid or not. A naive way would be to scan through the entire training set to validate the move. However, this approach is too slow involving a lot of unnecessary computation. A feasible alternative for larger datasets is to keep track of only a limited set of *marginal* points, i.e., $\mathcal{M} = \{i \in \mathcal{A} \mid -\xi_1 < \boldsymbol{\beta}_i < \xi_1\} \cup \{i \notin \mathcal{A} \mid -\xi_2 < |\mathbf{g}_i| - \lambda < \xi_2\}$ for small ξ_1 and ξ_2, and then discard all other points. The marginal points are those that are likely to join or leave the active set. Keeping track of only these points between two consecutive events makes the algorithm significantly more efficient. Every time after an event occurs, the set of marginal points will be updated accordingly.

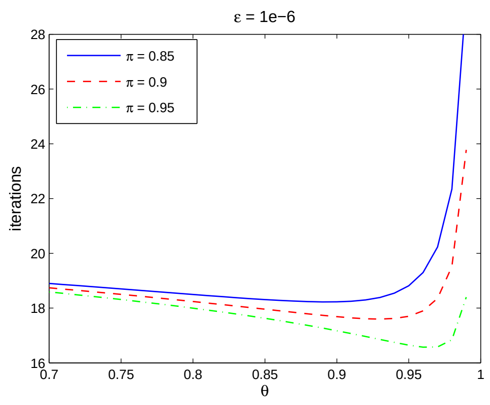


Figure 2: Number of iterations $\log_\theta \pi + \log_2(\log_{1-\epsilon}\theta)$ vs. decay rate θ. Three different π values (0.85, 0.9, 0.95) are considered and the error tolerance ϵ is set to 10^{-6}.

Since γ decreases at each iteration, the kernel matrix also changes accordingly. However, it is not necessary to re-compute the entire kernel matrix. To update the solution through equation (21), only the entries $K_\gamma(\mathbf{x}_i, \mathbf{x}_j)$ for $i, j \in \mathcal{A}$ need to be computed again. The cost of inverting an $m \times m$ matrix is $O(m^3)$. To monitor whether a marginal point $i \in \mathcal{M}$ joins or leaves the active set, the entries $K_\gamma(\mathbf{x}_i, \mathbf{x}_j), i \in \mathcal{M}, j = 1, ..., n$ need to be re-calculated and the conditions (12) and (13) are checked. Suppose there are p marginal points. Then $n \times p$ entries in the kernel matrix have to be updated. This leads to an overall complexity of $O(np + m^3)$ for each iteration. After one event occurs, the algorithm has to calculate the new $\mathbf{g}$ with a complexity of $O(n^2 m)$. The total number of events depends on both the range $[\gamma_{min}, \gamma_0]$ and the λ value specified by the user. If λ is large, only a small number of points will join or leave the active set during the algorithm, and vice versa. From the experiments, we notice that the number of events is always between $[0.5n, 5n]$.

4 Discussions

Note that the kernel function used in the algorithm can be very general. For example, we may use the polynomial kernel $K_{c,d}(\mathbf{x}_i, \mathbf{x}_j) = (\langle \mathbf{x}_i, \mathbf{x}_j \rangle + c)^d$ or the RBF kernel $K_\sigma(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/\sigma)$. If there exist multiple hyperparameters in the kernel function, the kernel solution path may still be traced in a multivariate space by updating the solution for one hyperparameter while holding the others fixed at each iteration. A requirement of the kernel path algorithm is that the matrix $\mathbf{A}$ is of full rank, which is equivalent to requiring that $\mathbf{H}^\gamma_{\mathcal{A}^l}$ is of full rank. This requirement can be satisfied by eliminating identical points and using the RBF kernel. However, this usually cannot be achieved by using other kernel functions. Fortunately, this problem can be alleviated by adding a small positive constant ν, called the ridge term, to each diagonal entry of the matrix $\mathbf{H}^\gamma_{\mathcal{A}^l}$, i.e.,

$$\mathbf{A} = \begin{bmatrix} 1 & \mathbf{1}^T \mathbf{K}^\gamma_{\mathcal{A}^l}/n \\ (\mathbf{K}^\gamma_{\mathcal{A}^l})^T \mathbf{1} & \mathbf{H}^\gamma_{\mathcal{A}^l} + \nu \mathbf{I} \end{bmatrix}. \tag{22}$$

This is in fact equivalent to adding an additional L2 regularizer with a small regularization parameter to the regularized empirical loss L_k. Thus, the vector $\mathbf{b}^a$ also needs to be modified as

$$\begin{bmatrix} -\beta_0^l - (\mathbf{K}^\gamma_{\mathcal{A}^l} \boldsymbol{\beta}^l_{\mathcal{A}^l})^T \mathbf{1}/n + \mathbf{y}^T \mathbf{1}/n \\ -\beta_0^l (\mathbf{K}^\gamma_{\mathcal{A}^l})^T \mathbf{1} - (\mathbf{H}^\gamma_{\mathcal{A}^l} + \nu \mathbf{I}) \boldsymbol{\beta}^l_{\mathcal{A}} + (\mathbf{K}^\gamma_{\mathcal{A}^l})^T \mathbf{y} - \mathrm{sgn}(\boldsymbol{\beta}^l_{\mathcal{A}^l})\lambda \end{bmatrix}. \tag{23}$$

With this modification, the kernel path algorithm can be applied to most kernel functions.

Rosset (2004) proposed a general path following algorithm based on second-order approximation when the solution path is not piecewise linear. It assumes that the solution update is not exact. Thus the algorithm only takes a very small step s in each iteration and applies a single Newton-Raphson step to approximate the next solution. Since it does not try to calculate the breakpoint value, the difference between the estimated value γ and the real breakpoint value γ^l is $|\gamma - \gamma_l| < s$. As a result, it is infeasible to keep the error tolerance to be very small. Otherwise, the total number of iterations will become very large. On the contrary, in our algorithm, if the number of iterations increases, the error tolerance will decrease exponentially, making it possible to keep the error tolerance arbitrarily small.

Since the optimal hyperparameter values are data dependent, the user has to choose the hyperparameter values carefully for different datesets. Generally speaking, there is a region in the hyperparameter space where the kLASSO model fits the data well. Outside this region either overfitting or underfitting is expected. Figure 3 is a hypothetical illustration of this generally observed phenomenon. We consider the RBF kernel $K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/\sigma)$ here and the two hyperparameters λ and σ are constrained to be positive. Given a data set, we assume there is a region $\lambda = [\lambda_1, \lambda_2]$ and $\sigma = [\sigma_1, \sigma_2]$ where the model performs well. When $\sigma > \sigma_2$, the regression function is not flexible enough and it always underfits the data. However, when $\sigma < \sigma_1$, the function is too elastic and is sensitive to most points. As a result, it always overfits the data. This provides us some heuristics to coordinate the two solution path algorithms when exploring the hyperparameter space.

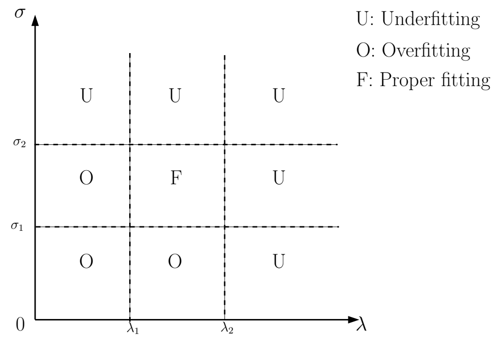


Figure 3: Performance of the model depends on the choice of the hyperparameter value.

5 Experiments

The behavior of solution path algorithms can best be illustrated using video. We have prepared some illustrative examples as video in `http://www.cse.ust.hk/~wanggang/sol_path/klasso.htm`.

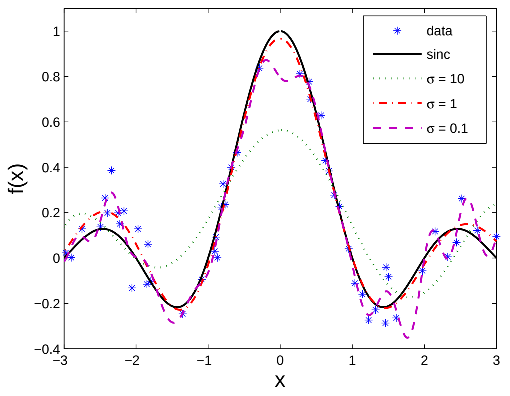


Figure 4: Based on three λ-paths with $\sigma = 10, 1, 0.1$, the optimal solution for each path in terms of the MSE on the validation set is plotted.

We randomly generate a set of 100 data points $\{(x_i, y_i)\}$ with x_i drawn uniformly from $[-3, 3]$ and $y_i = \sin(\pi x_i)/(\pi x_i) + e_i$, where e_i is a Gaussian noise term with mean zero and standard deviation 0.08. We randomly partition the data set into a training set of 50 points and a validation set of 50 points. The RBF kernel is used in our experiments, and hence the kernel path is traced with respect to σ.

We first consider the λ-path algorithm in kLASSO. The algorithm terminates when λ is less than 10^{-4}. Decreasing λ further does not help to further reduce L_k much but brings more redundant points into the active set. For each solution path, we compute the mean squared error (MSE) on the validation set for every regression function solution along the path. The solution that minimizes the MSE is then chosen and the corresponding regression function is plotted in Figure 4. The optimal regression function overfits the data when $\sigma = 0.1$ but underfits the data when $\sigma = 10$. It fits the data well when $\sigma = 1$. Figure 5(a) shows the MSE curves along the λ-paths for different kernel hyperparameter values. It is apparent that setting $\sigma = 10$ is too large and setting $\sigma = 0.1$ is too small to fit this data set of 50 points. The MSE on the validation set is minimized when an appropriate kernel parameter, $\sigma = 1$, is chosen. In Figure 5(b), we plot the active set size against λ for different values of σ. When $\sigma = 0.05$, the function is too elastic. The active set size generally increases as λ decreases. During this process, more and more points enter the active set and then settle down there. The regression function is thus sensitive to many points, leading to overfitting of the data. When $\sigma = 5$, on the other hand, the active set size always remains small. Since the function is not flexible enough, most points are not likely to stay inside the active set simultaneously. This leads to underfitting of the data. Hence, the active set size is one indicator of the generalization ability of the function. Figure 5(c) shows that λ decreases rapidly during the first few steps of the λ-path algorithm. Afterwards, the rate of decrease slows down significantly. Thus, the minimum MSE can be reached quite efficiently after only a small number of steps in the λ-path algorithm, provided that an appropriate kernel parameter is specified in advance.

We now consider tracing the kernel path, which is shifted from an intermediate step of the λ-path algorithm with $\sigma = 10$. As λ decreases, we select three different shifting points, $\lambda = 1, 0.1, 0.01$, to launch the kernel path and the kernel hyperparameter decreases from 10 to 0.1. The experimental results are shown in Figure 6. When λ is set to a large value like $\lambda = 1$, the regression function always underfits the data regardless of what the kernel hyperparameter value is. Thus, the MSE is large during the entire kernel path. In this case, we notice that only very few points enter the active set and hence the regression function cannot be represented well. The fact that the active set size is always small makes the kernel path algorithm terminate after a small number of steps. As the shifting point is set to $\lambda = 0.1$, a good regression function is found along the kernel path. The MSE is minimized as σ decreases to 1. However, decreasing σ further brings more points into the active set, making the MSE increase due to overfitting. Since more points are included in the active set, the number of steps required for the kernel path algorithm also increases. A good regression function is also found when the kernel path starts with $\lambda = 0.01$. However, the active size size becomes larger and the kernel path has to pass through more breakpoints. It indicates that we should stop exploring the hyperparameter space when the active set is already too large, since it costs unnecessary computation and overfitting is expected.

6 Conclusion

In this paper, we propose a novel method for tracing exactly the entire kernel path with respect to the kernel hyperparameter of the kLASSO regression model, even though the path is not piecewise linear. Due to the sparseness property of the L_1 regularizer in the regression function, the entire solution path can be traced very efficiently. Our method thus provides an attractive approach to the learning of the kernel parameter without having to train the model multiple times.

References

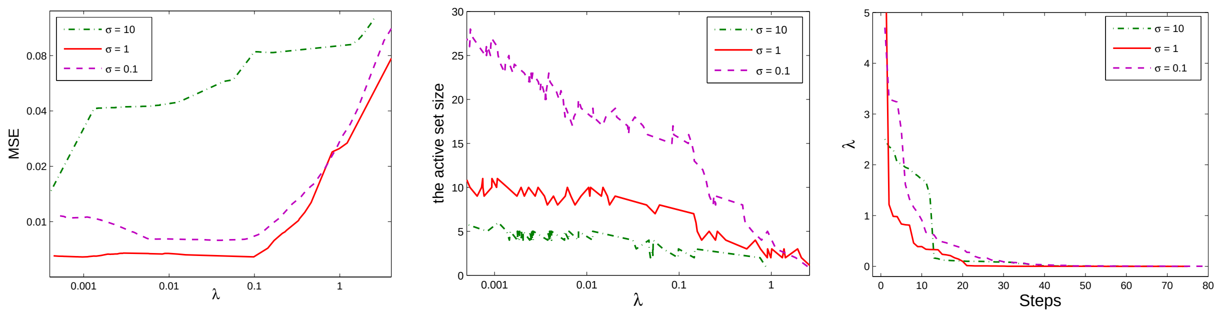
Bach, F., Thibaux, R., & Jordan, M. (2005). Regularization paths for learning multiple kernels. *Advances in Neural Information Processing Systems 17 (NIPS-05).*

Cristianini, N., Kandola, J., Elissee, A., & Shawe-Taylor, J. (2002). On kernel target alignment. *Advances in Neural Information Processing Systems 15 (NIPS-02).*

Efron, B., Hastie, T., Johnstone, I., & Tibshirani, R. (2004). Least angle regression. *The Annals of Statistics*, *32*, 407499.

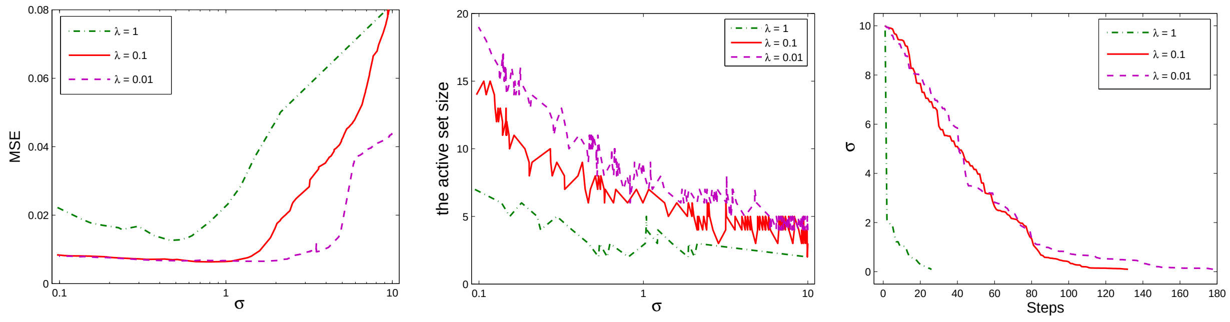
Hastie, T., Rosset, S., Tibshirani, R., & Zhu, J. (2004). The entire regularization path for the support vector machine. *Journal of Machine Learning Research*, *5*, 1391–1415.

Lanckriet, G., Cristianini, N., Bartlett, P., Ghaoui, L., & Jordan, M. (2004). Learning the kernel matrix



(a) (b) (c)

Figure 5: Relationship between MSE, λ, active set size and number of steps in the λ-paths for three different parameter values of the RBF kernel. (a) MSE vs. λ; (b) active set size vs. λ; (c) λ vs. number of steps. The horizontal axis in (a) and (b) is in log scale.



(a) (b) (c)

Figure 6: Relationship between MSE, γ, active set size and number of steps in the kernel paths for three different λ values. (a) MSE vs. σ; (b) active set size vs. σ; (c) σ vs. number of steps. The horizontal axis in (a) and (b) is in log scale.

with semidefinite programming. *Journal of Machine Learning Research*, *5*, 27–72.

Müller, K., Mika, S., Rätsch, G., Tsuda, K., & Schölkopf, B. (2001). An introduction to kernel-based learning algorithms. *IEEE Transaction on Neural Network*, *12*, 181–202.

Ng, A. Y. (2004). Feature selection, L_1 vs. L_2 regularization, and rotational invariance. *Proceedings of the 21th International Conference on Machine Learning (ICML-04).*

Ong, C., Smola, A., & Williamson, R. (2005). Learning the kernel with hyperkernels. *Journal of Machine Learning Research*, *6*, 1043–1071.

Rosset, S. (2004). Following curved regularized optimization solution paths. *Advances in Neural Information Processing Systems 17 (NIPS-04).*

Rosset, S., & Zhu, J. (2003). *Piecewise linear regularized solution paths* (Technical Report). Stanford University.

Roth, V. (2004). The generalized LASSO. *IEEE Transactions on Neural Networks*, *15*, 16–28.

Schölkopf, B., & Smola, A. (2002). *Learning with kernels.* MIT Press.

Tibshirani, R. (1996). Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society. Series B*, *58*, 267–288.

Wang, G., Yeung, D., & Lochovsky, F. (2006). Two-dimensional solution path for support vector regression. *Proceedings of the 23th International Conference on Machine Learning (ICML-06).*

Zhang, Z., Kwok, J., & Yeung, D. (2006). Model-based transductive learning of the kernel matrix. *Machine Learning*, *63*, 69–101.

Zhu, J., Rosset, S., Hastie, T., & Tibshirani, R. (2003). 1-norm support vector machines. *Advances in Neural Information Processing Systems 16 (NIPS-03).*