A Kernel Path Algorithm for Support Vector Machines

Gang Wang WANGGANG@CSE.UST.HK
Dit-Yan Yeung DYYEUNG@CSE.UST.HK
Frederick H. Lochovsky FRED@CSE.UST.HK
Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong, China

Abstract

The choice of the kernel function which determines the mapping between the input space and the feature space is of crucial importance to kernel methods. The past few years have seen many efforts in learning either the kernel function or the kernel matrix. In this paper, we address this model selection issue by learning the hyperparameter of the kernel function for a support vector machine (SVM). We trace the solution path with respect to the kernel hyperparameter without having to train the model multiple times. Given a kernel hyperparameter value and the optimal solution obtained for that value, we find that the solutions of the neighborhood hyperparameters can be calculated exactly. However, the solution path does not exhibit piecewise linearity and extends nonlinearly. As a result, the breakpoints cannot be computed in advance. We propose a method to approximate the breakpoints. Our method is both efficient and general in the sense that it can be applied to many kernel functions in common use.

1. Introduction

Kernel methods (Müller et al., 2001; Schölkopf & Smola, 2002) have demonstrated great success in solving many machine learning and pattern recognition problems. These 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. 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. In general, the input space does not have to be a vector space and hence structured, non-vectorial data can also be handled in the same way.

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. Ong et al. (2005) shows that the kernel function is a linear combination of a finite number of prespecified hyperkernel evaluations, and introduces a method to learn the kernel function directly in an inductive setting. Some approaches have been proposed (Cristianini et al., 2002; Bach et al., 2004; Lanckriet et al., 2004; Sonnenburg et al., 2006; Zhang et al., 2006) to seek a kernel matrix directly or learn a conic combination of kernel matrices from data.

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 m-fold cross validation to determine the best choice among a number of prespecified candidate hyperparameter values. Extensive exploration such as performing line search for one hyperparameter or grid search for two hyperparameters is usually applied. 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 large. Keerthi et al. (2006) proposed a hyperparameter tuning approach based on minimizing a smooth performance validation function. The direction to update the hyperparameters is the (negative) gradient of the validation func-

Appearing in *Proceedings of the 24th International Conference on Machine Learning*, Corvallis, OR, 2007.

tion with respect to the hyperparameters. This approach also requires training the model and computing the gradient multiple times.

Some approaches have been proposed to overcome these problems. A promising recent approach is based on solution path algorithms, which can trace the entire solution path as a function of the hyperparameter without having to train the model multiple times. A solution path algorithm is also called a regularization path algorithm if the hyperparameter involved in path tracing is the regularization parameter in the optimization problem. Efron et al. (2004) proposed an algorithm called the least angle regression (LARS). It can be applied to trace the regularization path for linear least square regression problem regularized with the L_1 norm. The path is piecewise linear and hence it is efficient to explore the entire path just by monitoring the breakpoints between the linear segments 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 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 regularization path and hence the entire path can be computed efficiently. 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 nonlinear regularization path. Besides the regularization parameter, we showed in our previous work (Wang et al., 2006) that this approach can also be applied to explore the solution paths for some other model hyperparameters.

In this paper, we propose a novel method that traces the solution path with respect to the kernel hyperparameter in SVC. We refer to this solution path as a *kernel path*. Given a kernel hyperparameter value and the optimal solution obtained for that value, we find that the solutions of the neighborhood hyperparameters can be calculated exactly. Since the kernel hyperparameter is embedded into each entry of the kernel matrix, the path is piecewise smooth but not piecewise linear. The implication is that the next breakpoint cannot be computed in advance before reaching it like what other solution path algorithms do. We propose a method to approximate the breakpoints. Unlike the path following algorithm of Rosset (2004) for nonlinear regularization paths, our approach is more efficient so that the kernel path can be traced efficiently. Moreover, our algorithm is general in the sense that it can be applied to many kernel functions in common use.

2. Problem Formulation

In a binary classification problem, we have a set of training pairs $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ where $\mathbf{x}_i \in \mathcal{X} \subseteq \mathcal{R}^d$ is the input data, $\mathcal{X}$ is the input space, and $y_i \in \{-1, +1\}$ is the class label. The goal is to learn a decision function $f(\mathbf{x})$ that can classify the unseen data as accurately as possible. Its associated classifier is $\text{sign}[f(\mathbf{x})]$. Using the *hinge loss*, the primal optimization problem for SVC can be formulated based on the standard regularization framework as follows:

$$\min_{f\in\mathcal{H}} \quad R_{primal} = \sum_{i=1}^n \xi_i + \frac{\lambda}{2}\|f\|_K^2 \tag{1}$$

$$\text{s.t.} \qquad y_i f(\mathbf{x}_i) \geq 1 - \xi_i \tag{2}$$

$$\xi_i \geq 0. \tag{3}$$

Here and below, $i = 1, ..., n$. $\|\cdot\|_K$ denotes the norm in the reproducing kernel Hilbert space (RKHS) $\mathcal{H}$ corresponding to a positive definite kernel function K. λ is the regularization parameter which gives the balance between the two opposing components of the objective function. A kernel function, i.e., $K_\gamma(\mathbf{x}_i, \mathbf{x}_j)$, is a bivariate function with its two independent variables defined in the input space. Here we explicitly show the hyperparameter γ of the kernel function in the subscript. Different values of γ embed the data into different feature spaces. In SVC, both λ and γ are hyperparameters in the model.

Using Wahba's representer theorem and Wolfe's duality theorem, we can derive the dual form of the optimization problem as:

$$\max_{\boldsymbol{\beta},\beta_0} \quad R_{dual}(\boldsymbol{\beta}, \beta_0) = \lambda \sum_{i=1}^n \beta_i - \frac{1}{2}\sum_{i,j=1}^n \beta_i\beta_j y_i y_j K_\gamma(\mathbf{x}_i, \mathbf{x}_j) \tag{4}$$

$$\text{s.t.} \qquad \sum_{i=1}^n y_i\beta_i = 0 \tag{5}$$

$$0 \leq \beta_i \leq 1, \tag{6}$$

where the dual variables (Lagrange multipliers) $\boldsymbol{\beta}$ are introduced for constraint (2). The decision function is given by

$$f(\mathbf{x}) = \frac{1}{\lambda}\left(\sum_{i=1}^n \beta_i y_i K_\gamma(\mathbf{x}, \mathbf{x}_i) + \beta_0\right). \tag{7}$$

$f(\mathbf{x})$ is expressed as an expansion in terms of only a subset of data points, called support vectors (SV), for which β_i is nonzero.

From the KKT optimality conditions, we have three cases to consider depending on the value of $y_i f(\mathbf{x}_i)$, which in turn determines how much the loss ξ_i is:

- If $y_i f(\mathbf{x}_i) > 1$, then $\xi_i = 0$ and $\beta_i = 0$;
- If $y_i f(\mathbf{x}_i) = 1$, then $\xi_i = 0$ and $\beta_i \in [0, 1]$;
- If $y_i f(\mathbf{x}_i) < 1$, then $\xi_i > 0$ and $\beta_i = 1$.

These three cases refer to points lying outside, at and inside the margins, respectively. We can see that β_i can take non-extreme values other than 0 and 1 at the optimal solution $\hat{\boldsymbol{\beta}}$ only if the value of $y_i f(\mathbf{x}_i)$ is equal to 1. β_0 is computed based on the points at the margins.

The optimal solution may be regarded as a vector function of some hyperparameter, denoted as μ:[1]

$$(\hat{\boldsymbol{\beta}}(\mu), \hat{\beta}_0(\mu)) = \arg\max_{(\boldsymbol{\beta}, \beta_0)} R_{dual}(\boldsymbol{\beta}, \beta_0; \mu). \tag{8}$$

Thus, the optimal solution varies as μ changes. For notational simplicity, we remove the $\hat{}$ notation in the rest of the paper. For every value of μ, we can partition the set of indices for the n data points into three subsets, $\mathcal{E}$, $\mathcal{L}$ and $\mathcal{R}$, which are referred to as the *elbow*, *left of the elbow*, and *right of the elbow*, respectively, to be consistent with the convention introduced in Hastie et al. (2004):

$$\mathcal{E} = \left\{ i : y_i \left(\sum_{j=1}^{n} \beta_j y_j K_\gamma(\mathbf{x}_i, \mathbf{x}_j) + \beta_0 \right) = \lambda, 0 \le \beta_i \le 1 \right\} \tag{9}$$

$$\mathcal{L} = \left\{ i : y_i \left(\sum_{j=1}^{n} \beta_j y_j K_\gamma(\mathbf{x}_i, \mathbf{x}_j) + \beta_0 \right) < \lambda, \beta_i = 1 \right\} \tag{10}$$

$$\mathcal{R} = \left\{ i : y_i \left(\sum_{j=1}^{n} \beta_j y_j K_\gamma(\mathbf{x}_i, \mathbf{x}_j) + \beta_0 \right) > \lambda, \beta_i = 0 \right\} \tag{11}$$

Therefore, the direction $\left(\frac{\partial \boldsymbol{\beta}(\mu)}{\partial \mu}, \frac{\partial \beta_0(\mu)}{\partial \mu} \right)$ in which the solution moves as μ changes should maintain the conditions (9)–(11) above.

Suppose $\mathcal{E}$ contains m indices which are represented as an m-tuple $(\mathcal{E}(1), \cdots, \mathcal{E}(m))$ such that $\mathcal{E}(i) < \mathcal{E}(j)$ for $i < j$. Let $\mathbf{y}_\mathcal{E} = (y_{\mathcal{E}(1)}, \ldots, y_{\mathcal{E}(m)})^T$, $\boldsymbol{\beta}_\mathcal{E} = (\beta_{\mathcal{E}(1)}, \ldots, \beta_{\mathcal{E}(m)})^T$, and $\boldsymbol{\beta}^a_\mathcal{E} = (\beta_0, \boldsymbol{\beta}^T_\mathcal{E})^T$. Equation (9) gives a linear system

$$\mathbf{L}(\boldsymbol{\beta}^a_\mathcal{E}(\mu), \mu) = 0 \tag{12}$$

with m linear equations. If now μ increases by an infinitesimally small step ϵ such that the three point sets remain unchanged and the conditions (9)–(11) still hold, then the corresponding linear system becomes $\mathbf{L}(\boldsymbol{\beta}^a_\mathcal{E}(\mu + \epsilon), \mu + \epsilon) = 0$.

Expanding $\mathbf{L}(\boldsymbol{\beta}^a_\mathcal{E}(\mu + \epsilon), \mu + \epsilon)$ via first-order Taylor series approximation around $\boldsymbol{\beta}^a_\mathcal{E}(\mu)$, we have

$$\begin{aligned} \mathbf{L}(\boldsymbol{\beta}^a_\mathcal{E}(\mu + \epsilon), \mu + \epsilon) &\simeq \mathbf{L}(\boldsymbol{\beta}^a_\mathcal{E}(\mu), \mu + \epsilon) \\ &+ \frac{\partial \mathbf{L}(\boldsymbol{\beta}^a_\mathcal{E}(\mu), \mu + \epsilon)}{\partial \boldsymbol{\beta}^a_\mathcal{E}} \left[\boldsymbol{\beta}^a_\mathcal{E}(\mu + \epsilon) - \boldsymbol{\beta}^a_\mathcal{E}(\mu) \right]. \end{aligned} \tag{13}$$

The gradient of $\mathbf{L}$ with respect to $\boldsymbol{\beta}^a_\mathcal{E}$ is a matrix of size $m \times (m + 1)$:

$$\frac{\partial \mathbf{L}}{\partial \boldsymbol{\beta}^a_\mathcal{E}} = \left[\mathbf{y}_\mathcal{E} \;\; \mathbf{K}_\gamma \right], \tag{14}$$

where $\mathbf{K}_\gamma = \left[y_{\mathcal{E}(i)} y_{\mathcal{E}(j)} K_\gamma(\mathbf{x}_{\mathcal{E}(i)}, \mathbf{x}_{\mathcal{E}(j)}) \right]_{i,j=1}^{m}$. It is interesting to note that the gradient $\partial \mathbf{L} / \partial \boldsymbol{\beta}^a_\mathcal{E}$ does not contain the parameter $\boldsymbol{\beta}^a_\mathcal{E}$ any more. Hence all terms after the first-order term of the Taylor series are in fact equal to zero, implying that the $\simeq$ in (13) should be replaced by $=$. From constraint (5), we also have

$$\mathbf{y}^T_\mathcal{E} (\boldsymbol{\beta}_\mathcal{E}(\mu + \epsilon) - \boldsymbol{\beta}_\mathcal{E}(\mu)) = 0. \tag{15}$$

Integrating equations (13) and (15) together, we know the next solution $\boldsymbol{\beta}^a_\mathcal{E}(\mu + \epsilon)$ can be updated as

$$\boldsymbol{\beta}^a_\mathcal{E}(\mu+\epsilon) = \boldsymbol{\beta}^a_\mathcal{E}(\mu) + \begin{bmatrix} 0 & \mathbf{y}^T_\mathcal{E} \\ \frac{\partial \mathbf{L}(\boldsymbol{\beta}^a_\mathcal{E}(\mu), \mu+\epsilon)}{\partial \boldsymbol{\beta}^a_\mathcal{E}} \end{bmatrix}^{-1} \begin{pmatrix} 0 \\ -\mathbf{L}(\boldsymbol{\beta}^a_\mathcal{E}(\mu), \mu + \epsilon) \end{pmatrix}. \tag{16}$$

Hence, given a hyperparameter μ and its corresponding optimal solution $(\boldsymbol{\beta}(\mu), \beta_0(\mu))$, the solutions of its neighborhood hyperparameters can be computed exactly as long as the three point sets remain unchanged. However, some events may occur when we change the value of μ to a larger extent. An event is said to occur when some point sets change. We categorize these events as follows:

- A new point i joins $\mathcal{E}$, i.e., the condition (10) for a variable β_i with $i \in \mathcal{L}$ or the condition (11) for a variable β_i with $i \in \mathcal{R}$ ceases to hold if $\boldsymbol{\beta}^a_\mathcal{E}(\mu)$ keeps moving in the same direction.
- The variable β_i for some $i \in \mathcal{E}$ reaches 0 or 1. In this case, the condition (9) will cease to hold if $\boldsymbol{\beta}^a_\mathcal{E}(\mu)$ changes further in the same direction. Thus, point i leaves $\mathcal{E}$ and joins some other set.

By monitoring the occurrence of these events, we can find the next breakpoint at which the updating formula needs to be calculated again. The algorithm then updates the point sets and continues to trace the path.

[1] μ is very general. It can be used to represent any hyperparameter such as λ or γ in SVC.

2.1. Regularization Path Algorithm

Our goal is to generate the solution path for a range of hyperparameter values by repeatedly calculating the next optimal solution based on the current one. Based on the updating formula in equation (16), we can easily derive the path following algorithm with respect to the regularization parameter λ. Replacing μ with λ, we have

$$\boldsymbol{\beta}^a_{\mathcal{E}}(\lambda+\epsilon) = \boldsymbol{\beta}^a_{\mathcal{E}}(\lambda) + \epsilon \begin{bmatrix} 0 & \mathbf{y}_{\mathcal{E}}^T \\ \mathbf{y}_{\mathcal{E}} & \mathbf{K}_\gamma \end{bmatrix}^{-1} \begin{pmatrix} 0 \\ \mathbf{1} \end{pmatrix}, \quad (17)$$

where the γ value is fixed. Note that equation (17) for the solution updating is equivalent to the formula in Hastie et al. (2004). Substituting (17) into (7), we can calculate the next breakpoint at which the conditions (9)–(11) will not hold if λ is changed further. As a result, the regularization path can be explored.

3. Kernel Path Algorithm

Replacing μ by γ in equation (16), the kernel path algorithm can be derived in a similar manner. We consider the period between the lth event (with $\gamma = \gamma^l$) and the $(l+1)$th event (with $\gamma = \gamma^{l+1}$). The sets $\mathcal{E}$, $\mathcal{L}$ and $\mathcal{R}$ remain unchanged during this period. Thus we trace the solution path of $(\boldsymbol{\beta}_{\mathcal{E}}(\gamma), \beta_0(\gamma))$ as γ changes.

Theorem 1 *Suppose the optimal solution is $(\boldsymbol{\beta}^l, \beta_0^l)$ when $\gamma = \gamma^l$. Then for any γ in $\gamma^{l+1} < \gamma < \gamma^l$, we have the following results:*

- *For the points $i \in \mathcal{L} \cup \mathcal{R}$, $\beta_i = \beta_i^l$ is fixed at 0 or 1, which is independent of γ;*
- *The solution to $(\boldsymbol{\beta}_{\mathcal{E}}, \beta_0)$ is given by*

$$\begin{pmatrix} \beta_0 \\ \boldsymbol{\beta}_{\mathcal{E}} \end{pmatrix} = \begin{pmatrix} \beta_0^l \\ \boldsymbol{\beta}_{\mathcal{E}}^l \end{pmatrix} + \begin{bmatrix} 0 & \mathbf{y}_{\mathcal{E}}^T \\ \mathbf{y}_{\mathcal{E}} & \mathbf{K}_\gamma \end{bmatrix}^{-1} \begin{pmatrix} 0 \\ \mathbf{b}_\gamma \end{pmatrix}, \quad (18)$$

where

$$\mathbf{K}_\gamma = \left[y_{\mathcal{E}(i)} y_{\mathcal{E}(j)} K_\gamma(\mathbf{x}_{\mathcal{E}(i)}, \mathbf{x}_{\mathcal{E}(j)}) \right]_{i,j=1}^m, \quad (19)$$

$$\mathbf{b}_\gamma = \left(-y_{\mathcal{E}(i)} \left[\sum_{j=1}^n \beta_j^l y_j K_\gamma(\mathbf{x}_{\mathcal{E}(i)}, \mathbf{x}_j) + \beta_0^l \right] + \lambda \right)_{i=1}^m \quad (20)$$

As such, we can update the solution to $(\boldsymbol{\beta}_{\mathcal{E}}, \beta_0)$ exactly while those to others remain unchanged. The value of γ can either increase or decrease. As γ changes, the algorithm monitors the occurrence of any of the following events:

- One of the $\beta_{\mathcal{E}(i)}$ for $i = 1, \ldots, m$ reaches 0 or 1.
- A point $i \notin \mathcal{E}$ hits the elbow, i.e., $y_i f(\mathbf{x}_i) = 1$.

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 by γ^{l+1}. We then update the point sets and continue until the algorithm terminates.

Table 1. Kernel path algorithm.

Input:
$\hat{\boldsymbol{\beta}}$, $\hat{\beta}_0$, γ_0 - initial solution for γ;
θ, ϵ, γ_{min} - decay rate, error tolerance, γ limit

```
1. t = 0; β^0 = β̂, β_0^0 = β̂_0
2. while γ > γ_min
3.     r = θ;
4.     while r < 1 − ε
5.         γ = rγ_t;
6.         use (18) to compute (β(γ), β_0(γ))
7.         if (β(γ), β_0(γ)) is the valid solution
8.             β^{t+1} = β(γ); β_0^{t+1} = β_0(γ); γ_{t+1} = γ;
               t = t + 1
9.         else r = r^{1/2}
10.        endif
11.    end-while
12.    update the point sets E, L, R;
13.end-while ;
```

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

In the previous works (Zhu et al., 2003; Hastie et al., 2004; Wang et al., 2006), the solution path is piecewise linear with respect to some 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 at $\gamma = \gamma^l$. We propose here an efficient algorithm for estimating the next breakpoint, i.e., γ^{l+1}, at which the next event occurs. Table 1 shows the pseudocode of our proposed kernel path algorithm. The user has to specify a decay rate $\theta \in (0, 1)$. 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 in advance by the user. Hence, we can estimate the breakpoint γ such that

$\gamma^{l+1}(1-\epsilon) \geq \gamma \geq \gamma^{l+1}$. Note that this algorithm only describes the kernel path algorithm in the decreasing direction. In general, γ can either increase or decrease. The only difference for the increasing direction is to set θ greater than 1. An SVM solver is needed in the beginning to initialize the optimal solution for some γ value.

We assume on average 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. Setting the error tolerance ϵ to a large value may lead to the algorithm getting stuck, and here we set it to 10^{-6}. It is not necessary to set ϵ to be very small, say 10^{-9}, which will require a great deal of unnecessary computation. Figure 1 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. If θ is chosen to be not very close to 1, then the number of iterations does not vary much. We set $\theta = 0.95$ and $\epsilon = 10^{-6}$ in our experiments, and the algorithm always takes less than 20 iterations to reach the next breakpoint.

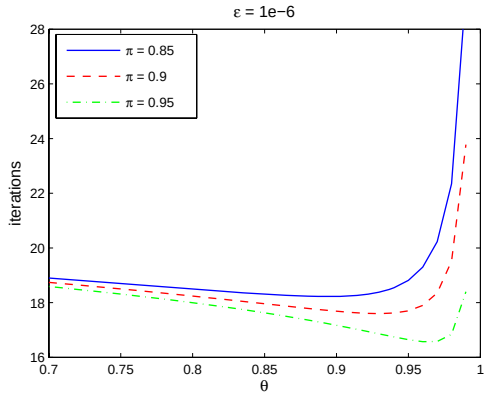


Figure 1. Number of iterations $\log_\theta \pi + \log_2(\log_{1-\epsilon} \theta)$ vs. decay rate θ. Three different π values (0.85, 0.9, 0.95) are considered and the error tolerance ϵ is set to 10^{-6}.

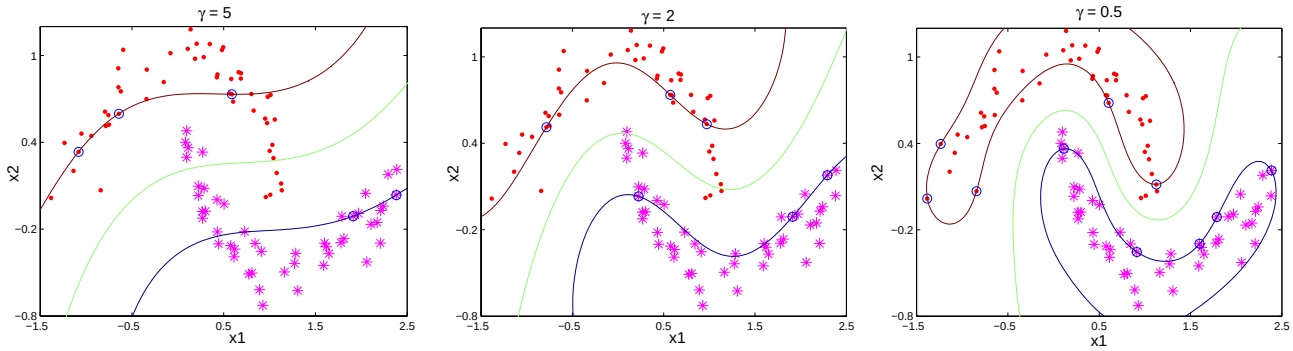
Since γ decreases at each iteration, the kernel matrix also changes accordingly. However, it is not necessary to recompute the entire kernel matrix. To update the solution through equation (18), only the entries $K_\gamma(\mathbf{x}_i, \mathbf{x}_j)$ for $i \in \mathcal{E}$ and $j \in \mathcal{E} \cup \mathcal{L}$ need to be computed again. The cost of calculating $\mathbf{b}_\gamma$ is $O(mn)$ and that of inverting an $(m+1)\times(m+1)$ matrix is $O(m^3)$. To monitor whether a point leaves the current set to join another set, the entries $K_\gamma(\mathbf{x}_i, \mathbf{x}_j), i \in 1, ..., n, j \in \mathcal{E} \cup \mathcal{L}$ need to be re-calculated in order to check the conditions (9)–(11). These lead to an overall complexity of $O(n^2 + m^3)$ for each iteration where m is typically much smaller than n. The total number of events is the number of times that the points pass through $\mathcal{E}$. Its value depends on both the range $[\gamma_{min}, \gamma_0]$ and the λ value specified by the user. From many experiments, empirical findings suggest that the number of iterations is always some small multiple c of n for a large enough range of γ, which implies that the total computational cost is $O(cn^3 + cnm^3)$. In practice, users usually try out a small number of hyperparameter values on an SVM solver to gain some basic understanding of the data. This preliminary investigation can provide some hints on how to specify the region in which the users are interested for the kernel path algorithm.

4. Discussion

In step 7, the algorithm checks whether the new solution is valid or not. A naive way is to scan through the entire training set to validate the move. However, this is too slow involving a great deal of unnecessary computation. A feasible alternative for larger datasets is to keep track of only a small set of *marginal* points, i.e., $\mathcal{M} = \{i \in \mathcal{E} \,|\, \beta_i < \xi_0 \text{ or } \beta_i > 1 - \xi_1\} \cup \{i \in \mathcal{L} \,|\, y_i f(\mathbf{x}_i) > 1 - \xi_2\} \cup \{i \in \mathcal{R} \,|\, y_i f(\mathbf{x}_i) < 1 + \xi_3\}$ for small positive numbers ξ_0, ξ_1, ξ_2 and ξ_3, and discard all other points. A marginal point is one that is likely to leave the set to which it currently belongs and join another set. Thus, only a small number of entries in the kernel matrix need to be re-calculated. Keeping track of only these points between two consecutive events makes the algorithm significantly more efficient. Every time after one or several events occur, the set of marginal points is then updated accordingly.

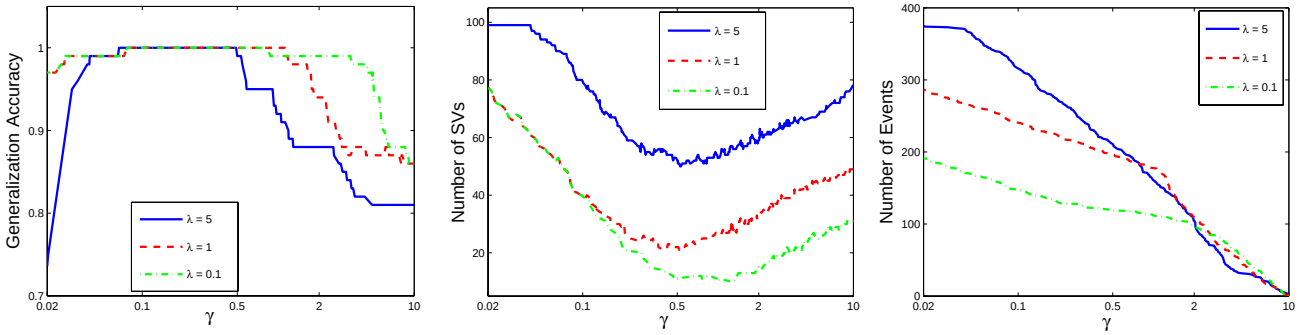
Note that the kernel function used in the algorithm can be very general. For example, we may use the polynomial kernel $K_{c,d}(\mathbf{x}_i, \mathbf{x}_j) = (\langle \mathbf{x}_i, \mathbf{x}_j \rangle + c)^d$ or the Gaussian RBF kernel $K_\gamma(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/\gamma)$. If the kernel function contains multiple hyperparameters, 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.

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



(a) $\gamma = 5$ (b) $\gamma = 2$ (c) $\gamma = 0.5$

Figure 2. SVM classification results of the two-moons dataset for different kernel hyperparameter values ($\gamma = 5, 2, 0.5$). In each sub-figure, the middle line (green) shows the decision boundary and the other two lines specify the margins with the points it contains indicated by blue circles. λ is set to 1.



(a) (b) (c)

Figure 3. Change in (a) the generalization accuracy and (b) the number of SVs and (c) the number of events occurred along the kernel path with different λ values for the two-moons dataset. The horizontal axis is in logarithm scale. The generalization accuracy first increases and then decreases and the number of SVs first decreases and then increases as γ decreases from 10 to 0.2.

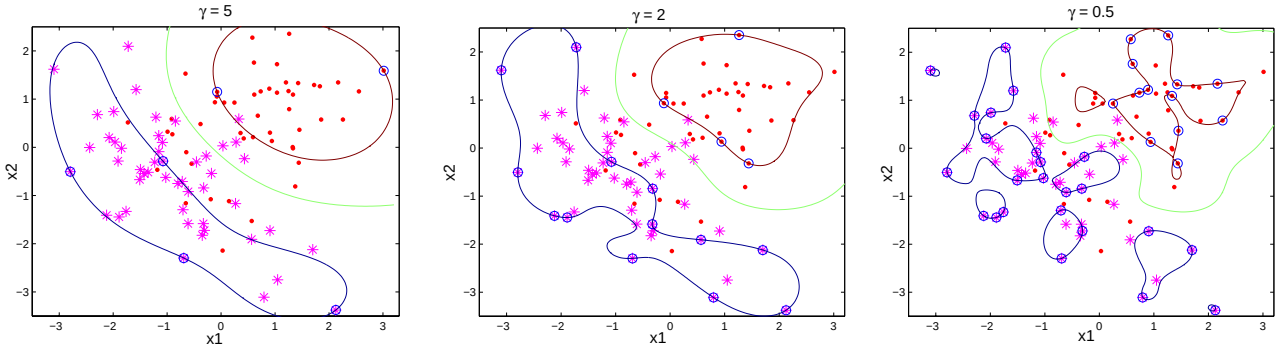
to be very small. Otherwise, the total number of iterations will become very large. On the contrary, in our algorithm, if the number of iterations increases, the error tolerance will decrease exponentially, making it possible to keep the error tolerance arbitrarily small.

5. Experimental Results

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

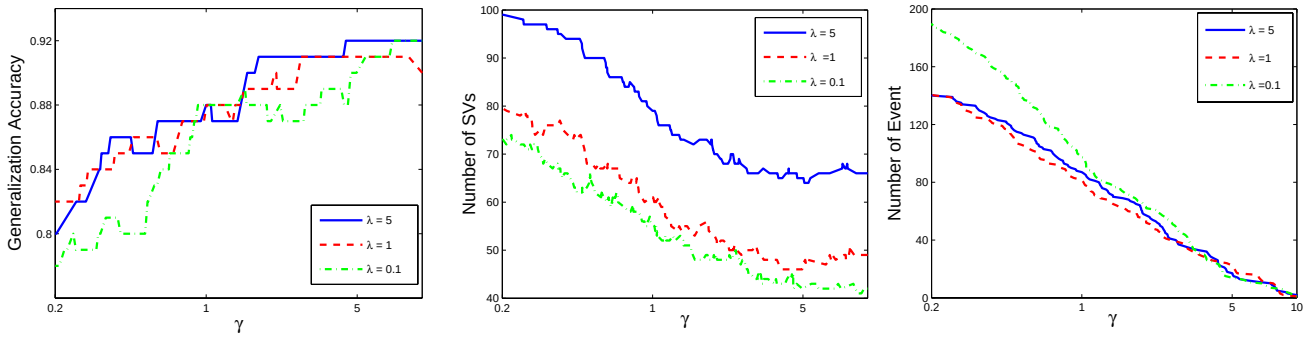
In our experiments, we consider two binary classification datasets which possess very different properties. The first is the two-moons dataset with strong manifold structure. It is generated according to a pattern of two intertwining moons and the data points in the two classes are well separated from each other. The second dataset is generated from two Gaussian distributions with different means and covariances corresponding to the positive and negative classes which partially overlap with each other. Each of the two datasets contains 100 positive points and 100 negative points. We randomly select 50% of the points for training and the rest for estimating the generalization error. The Gaussian RBF kernel is used in the experiments. To explore the solutions along the kernel path, we first use LIBSVM (Chang & Lin, 2001) to compute an initial optimal solution and then execute the kernel path algorithm as described above.

For the two-moons dataset, the kernel path starts from $\gamma = 10$ and extends in the decreasing direction of γ until $\gamma = 0.02$, while setting $\lambda = 1$. Figure 2 shows the classification results at three points of the kernel



(a) $\gamma = 5$ (b) $\gamma = 2$ (c) $\gamma = 0.5$

Figure 4. SVM classification results of the two-Gaussians dataset for different kernel hyperparameter values ($\gamma = 5, 2, 0.5$). λ is set to 1.



(a) (b) (c)

Figure 5. Change in (a) the generalization accuracy and (b) the number of SVs and (c) the number of events occurred along the kernel path with different λ values for the two-Gaussians dataset. The horizontal axis is in logarithm scale. The generalization accuracy decreases and the number of SVs increases as γ decreases from 10 to 0.2.

path, $\gamma = 5, 2, 0.5$. As we can see, different kernel hyperparameter values produce dramatically different decision boundaries. When γ is large, the decision boundary is closer to linear, which leads to large generalization error in this dataset. As γ decreases, the decision boundary becomes more flexible to fit into the gap between the two classes. Thus, the generalization error decreases accordingly. When $\gamma = 0.5$, the decision function well separates the points between the two classes and only a small number of SVs are used to represent the function. Figure 3 shows the change in the generalization accuracy, the number of SVs, and the number of events occurred along the kernel path with different λ values. We notice that the generalization accuracy increases and the number of SVs decreases as γ decreases from 10 to 0.5. Since there exists a well-separated boundary between the two classes, setting a smaller λ value will give fewer SVs and attain better performance. Although the generalization accuracies in the beginning of the kernel paths have significant differences, they all can reach 100% accuracy as γ decreases. However, as γ further decreases to be very small, the margins become so elastic that most of the input points are included as SVs. Hence, the decision function has high variance, which leads to the overfitting problem. We also notice from Figure 3(c) that the computational cost of the kernel path can be optimized by setting a proper regularization value.

Figure 4 shows the classification results of the two-Gaussians dataset for different γ values. Unlike the two-moons dataset, this dataset has an overlapping region between the two classes. Hence, the optimal decision boundary is expected to be a smooth surface that passes through the overlapping region. This can be obtained by setting γ to a relatively large value. Since setting a larger λ value in this dataset leads to wider margins and a smoother decision boundary, higher generalization accuracy can be obtained and

more points become SVs. As the kernel path extends in the decreasing direction, the decision function becomes more rugged and a higher degree of overfitting will occur. The decision boundary changes slightly as the kernel path starts from $\gamma = 10$ and both the number of SVs and the generalization accuracy remain stable. The decision function shows more significant changes as the kernel path algorithm proceeds after γ is less than 2. As γ decreases, more and more points become SVs and the generalization accuracy drops dramatically. Figure 5(a) and (b) illustrate this. As a smaller λ value tends to minimize the training error, more severe overfitting will occur along the kernel path for $\lambda = 0.1$ than for $\lambda = 5$ in this dataset. When γ becomes small, most points become SVs and the decision function is no longer a very sparse representation. Its generalization accuracy thus decreases.

6. Conclusion and Future Work

In kernel machines, it is very important to set the kernel hyperparameters to appropriate values in order to obtain a classifier that generalizes well. In this paper, we derive an exact updating formula to calculate solutions along the kernel path in SVC. Since the piecewise linearity property does not hold along the kernel path, breakpoints cannot be calculated in advance. We propose an efficient algorithm to find the breakpoints so that the kernel path can be traced without having to train the model multiple times.

It is also important to set a proper regularization value in SVC, since it can not only produce a well generalized decision function but can also reduce the number of iterations, leading to speedup of the kernel path algorithm. We are interested in an integration of this kernel path algorithm with a regularization path algorithm into a unified approach for tracing solution paths in a two-dimensional hyperparameter space. This direction will be explored in our future work.

Acknowledgment

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. The author would also like to thank the anonymous reviewers for their constructive comments.

References

Bach, F., Lanckriet, R., & Jordan:, M. (2004). Multiple kernel learning, conic duality, and the SMO algorithm. *Proceedings of the 21th International Conference on Machine Learning (ICML-04)*.

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

Chang, C., & Lin, C. (2001). *LIBSVM: a library for support vector machines*. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`.

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*, 407–499.

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.

Keerthi, S., Sindhwani, V., & Chapelle, O. (2006). An efficient method for gradient-based adaption of hyperparameters in SVM models. *Advances in Neural Information Processing Systems 19 (NIPS-06)*.

Lanckriet, G., Cristianini, N., Bartlett, P., Ghaoui, L., & Jordan, M. (2004). Learning the kernel matrix 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 Transactions on Neural Networks*, *12*, 181–202.

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.

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

Sonnenburg, S., Rätsch, G., Schäfer, C., & Schölkopf, B. (2006). Large scale multiple kernel learning. *Journal of Machine Learning Research*, *7*, 15311565.

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)*.