doi:10.3969/j.issn.1671-637X.2013.06.012

基于SVM的多变量函数回归分析研究

刘洛霞
(上海飞机客户服务有限公司,上海 200240)

摘 要:研究了通过自动调整回归因子多项参数方式提高基于SVM函数回归性能的问题。通过使用梯度下降法和遗传算法最小化SVM的归一化误差估计值来实现SVM函数回归性能的提高。大量的试验结果验证了所提出方法的性能。

关键词:梯度下降法;遗传算法;归一化误差估计;支持向量机(SVM)
中图分类号:V271.4 **文献标志码**:A **文章编号**:1671-637X(2013)06-0050-08

Choosing Multiple Parameters for Function Regression Based on SVM

LIU Luoxia
(Shanghai Aircraft Customer Service Co. Ltd., Shanghai 200240)

Abstract: Performance improvement of function regression based on Support Vector Mmachines (SVM) by automatically tuning multiple parameters of regressors was considered. This was done by minimizing some estimation of the generalization error of SVM using gradient descent and Genetic Algorithm (GA) over the set of parameters. Performance of the proposed method was illustrated by extensive experimental results.

Key words: gradient descent; genetic algorithm; normalized error estimation; Support Vector Machine(SVM)

0 Introduction

In many scientific and engineering research files, many problems such as model identification in control engineering, channel identification in communication engineering and density function in statistics, can be generalized to solve a multivariable function regression problem. In the problem of function regression: one takes a set basis functions (regressors) $g_i(X, \theta_i)$ and a finite set of input-output data pairs $Z = \{(X_1, \theta_1), \cdots, (X_l, \theta_l)\}$ and attempts to find out weight vector W to construct a function $f(X, W, \theta) = \sum_i w_i g_i(X, \theta_i)$. Such that some error criteria is minimized. There are many method proposed to deal with this problem, such as linear regression, RBF (Radial Basis Function) network, neural network, MARS (Multivariable Adaptive Regression Spline), projection pursuit and fuzzy system. If we know the type of function (linear, quadratic or gaussian, etc) with unknown parameters, this problem is called parametric regression. Otherwise it is called non-parametric regression (MARS and fuzzy clustering).

For each regression model, the question must be asked: how we can get the model, built on finite amount of data, capture the concept underlying the data after learning? This question is about the generalization performance, which is closely related to several well-known problems in the statistics and machine learning literature, such as the structural risk minimization (SRM)[1], the bias variance dilemma[2], and the over-fitting phenomena[3]. Loosely speaking, a model, built up on finite amount of training data, generalizes the best if the right tradeoff is found between the training accuracy and the "capacity" of the model set from which the model is chosen. On the hand, a low "capacity" model set may not contain any model that fits the training data well. On the other hand, too much freedom may eventually generate a model behaving perfectly on training data and poorly on generalization.

In statistical learning literature, the Vapnik-Chervonenikis (VC) theory[4] provides the general measure of model set complexity, and gives associated bounds on generalization. SVM is directly theoretical product when VC theory is applied to classification problem (similar for regression

收稿日期:2011-09-30 修回日期:2012-04-16
作者简介:刘洛霞(1969—),女,河南洛阳人,硕士,研究方向为飞行控制。

problem). By assigning learning procedure to a quadratic programming to get global minimize on a convex set, SVM reduces not only empirical risk but also VC dimension to get good generalization performance.

The SVM algorithm usually depends on several parameters-learning parameters and kernel parameters. These parameters also affect the performance of SVM. We present here technique that allows to deal with a large number parameters thus to improve function regression performance.

1 VC theory and Support Vector Machines

For gentle tutorials, we refer interested readers to Burges[5] and Smola et al[6]. More exhaustive treatments can be found in the book by Vapnik[4].

1.1 VC Theory

Let's consider two-class classification problem of assigning class label $y = \{+1, -1\}$ to input vector $X \in \mathbf{R}^n$. We give a set of training samples $\{(\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_l, y_l)\} \subset \mathbf{R}^n \times \{+1, -1\}$ that are drawn independently from some unknown cumulative probability distribution $P(x, y)$. The learning task is formulated as finding a function $f: \mathbf{R}^n \to \{+1, -1\}$ that "best" approximates the mapping generating the training set. In order to make learning feasible, we need to specify a function H, from which a function is chosen.

An ideal measure of generalization performance for a selected function f is expected risk defined as $R_{(P(x,y))}(f) = \int_{\mathbf{R}^n \times \{+1, -1\}} I_{|f(X) \neq y|}(x, y)\,\mathrm{d}P(x, y)$. Where $I_A(z)$ is an indicator function such that $I_A(z) = 1$ for all $z \in A$, and $I_A(z) = 0$ for all $z \notin A$. Unfortunately, this is more an elegant way of writing the error probability than practical usefulness because $P(x, y)$ is usually unknown. However, there is family of bounds on the expected risk, which demonstrates fundamental principle of building function with good generalization. Here we present one result from the VC theory due to Vapnik and Chervonenkis: given a set of l training samples and function space H, with probability $1 - \eta$, for any $f \in H$ the expected risk is bounded above by

$$R_{P(x,y)}(f) \leqslant R_{\mathrm{emp}}(f) + \sqrt{\frac{h(1 + \frac{2l}{h}) - \ln\frac{\eta}{4}}{l}} \quad (1)$$

for any distribution $P(X, y)$ on $\mathbf{R}^n \times \{+1, -1\}$. Here $R_{\mathrm{emp}}(f)$ is called the empirical risk (or training error), h is a non-negative integer called the VC dimension. The VC dimension is a measure of the capacity of a $\{+1, -1\}$-valued function space. Given a training set of size l, (1) demonstrates a strategy to control expected risk by controlling two quantities: the empirical risk and the VC dimension. There are similar results for function regression.

1.2 Support Vector Machines

Let $\{(\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_l, y_l)\} \subset \mathbf{R}^n \times \{+1, -1\}$ be a training set. The SV approach tries to find a canonical hyperplane $\{X \in \mathbf{R}^n : \langle W, X \rangle + b = 0, W \in \mathbf{R}^n, b \in \mathbf{R}\}$, that maximally separates two classes of training samples. Here $\langle \cdot, \cdot \rangle$ is an inner product in $\mathbf{R}^n$. The corresponding decision function $f: \mathbf{R}^n \to \{+1, -1\}$ is then given by $f(X) = \mathrm{sgn}(\langle W, X \rangle + b)$. Consideration that the training set may not be linearly separable, the optimal decision function is found by solving the following quadratic program:

$$\text{minimize} \quad J(W, \boldsymbol{\xi}) = \frac{1}{2}[W, W] + C\sum_{i=1}^{l}\xi_i \quad (2)$$

$$\text{subject to} \quad y_i(\langle W, X_i \rangle + b) \geqslant 1 - \xi_i, \xi_i \geqslant 0, i = 1, \cdots, l,$$

where $\boldsymbol{\xi} = [\xi_1, \cdots, \xi_l]^{\mathrm{T}}$ are slack variables introduced to allow to for the possibility of misclassification of training samples, $C > 0$ is some constant.

In order to minimize (2) relate to our ultimate goal of optimizing the generalization, we need to introduce a theorem about VC dimension of canonical hyperplanes, which is stated as follows. For a given set of l training samples, let R be the radius of the smallest ball containing all l training samples, and $\Lambda \subset \mathbf{R}^n \times \mathbf{R}$ be the set of coefficients of canonical hyperplanes defined on the training set. The VC dimension h of the function space $H = \{f(X) = \mathrm{sgn}([W, X] + b) : (W, b) \in \Lambda, \| W \| \leqslant A, X \in \mathbf{R}^n\}$ is bounded above by $h \leqslant \min(R^2A^2, n) + 1$.

Thus to minimize in (2) amounts to minimize the VC dimension of the function space H, therefore the second term of the bound (1). On the other hand, $\sum_{i=1}^{l}\xi_i$ is upper bound on the number of misclassification on the training set, thus controls the empirical risk term in (1). For an adequate positive constant C, minimizing (2), can indeed decrease the upper bound on the expected risk.

Applying the Karush-Kuhn-Tucker complementarity condition, we can show that a W, which minimize (2), can be written as $W = \sum_{i=1}^{l} y_i\alpha_i X_i$. This is called the dual representation of W. An X_i with nonzero α_i is called support vector. Let S be the index set of support vectors, then the optimal decision function becomes

$$f(X) = \mathrm{sgn}(\sum_{i \in S} y_i\alpha_i\langle X, X_i \rangle + b) \quad (3)$$

Where the coefficients can be found by solving the dual problem of (2):

$$\text{maximize} \quad w(\boldsymbol{\alpha}) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{l} \alpha_i\alpha_j y_i y_j \langle X_i, X_j\rangle \quad (4)$$

$$\text{subject to} \quad C \geqslant \alpha_i \geqslant 0, i = 1, \cdots, l, \text{ and } \sum_{i=1}^{l} \alpha_i y_i = 0.$$

The decision boundary given by (3) is a hyperplane in $\mathbf{R}^n$. More complex decision surfaces can be generated by employing a nonlinear mapping $\phi: \mathbf{R}^n \to F$ to map the data into a new feature space F (usually has dimension higher than n), and solving the same optimization problem in F, i. e., find the maximal separating hyperplane in F. Note that in (4) never appears isolated but always in the form of inner product (X_i, X_j). This implies that there is no need to evaluate the nonlinear mapping as long as we know the inner product if any given $X, Z \in \mathbf{R}^n$. So for computational purposes, instead of defining $\Phi: \mathbf{R}^n \to F$ explicitly, a function $K: \mathbf{R}^n \times \mathbf{R}^n \to \mathbf{R}$ is introduced to directly define an inner product in F, i. e., $\boldsymbol{K}(X_i, X_j) = (\phi(X_i), \phi(X_j))_F$ where $(\cdot, \cdot)_F$ is an inner product in F, and ϕ is a nonlinear mapping induced by K. Such a function K is also called the Mercer kernel. Substituting $\boldsymbol{K}(X_i, X_j)$ for (X_i, X_j) in (4) produces a new optimization problem

$$\text{maximize} \quad w(\boldsymbol{\alpha}) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{l} \alpha_i\alpha_j y_i y_j \boldsymbol{K}(X_i, X_j) \quad (5)$$

$$\text{subject to} \quad C \geqslant \alpha_i \geqslant 0, i = 1, \cdots, l, \text{ and } \sum_{i=1}^{l} \alpha_i y_i = 0.$$

Solving it for $\boldsymbol{\alpha}$ gives a decision function of the form

$$f(X) = \operatorname{sgn}\left(\sum_{i \in S} y_i \alpha_i K(X, X_i) + b\right) \quad (6)$$

whose decision boundary is a hyperplane in F, and translates to nonlinear boundaries in the original space.

The SV approach can also be applied to regression problem by replace $\sum_{i=1}^{l} \xi_i$ term in (2) with a new loss term $\sum_{i=1}^{l} L^{\varepsilon}(X_i, y_i, f)$ and adjusting the constraints accordingly. $L^{\varepsilon}(X_i, y_i, f)$ is a linear ε-insensitive loss function defined as $L^{\varepsilon}(X_i, y_i, f) = (|y - f(X)| - \varepsilon) I_{\{|y-f(X)| \geqslant \varepsilon\}}(X, y, f)$, i. e., only errors falling outside the interval $[-\varepsilon, +\varepsilon]$ counts. It was shown (Critianini & Taylor, 2000) that the function minimizing (2) with the new loss term has a form[7]

$$f(Z) = \sum_{i=1}^{l} \alpha_i K(Z, X_i) + b \quad (7)$$

To find the coefficients we have to solve the following quadratic program

$$\max w(\boldsymbol{\alpha}) = \sum_{i=1}^{l} y_i\alpha_i - \varepsilon\sum_{i=1}^{l} |\alpha_i| - \frac{1}{2}\sum_{i,j=1}^{l} \alpha_i\alpha_j \boldsymbol{K}(X_i, X_j)$$

$$\text{subject to} \quad C \geqslant \alpha_i \geqslant -C, i = 1, \cdots, l, \text{ and } \sum_{i=1}^{l} \alpha_i = 0.$$

A detailed discussion on generalization performance of SV function regression can be found[6].

2 Gradient Descent Optimization Approach

2.1 General description

The SVM algorithm usually depends on several parameters: learning parameters (C and ε) and kernel parameters, which appear in kernel function $K(X, Z)$. We can change these parameters to get better performance (for time limit, only kernel parameters are considered). For example we will see how to use radial basis function kernels (RBF) with many different scaling factors (σ_i) as input dimensions: $K(X, Z) = \exp\left(-\sum_i \frac{(x_i - z_i)^2}{2\sigma_i^2}\right)$.

The usual approach is to consider $\boldsymbol{\sigma} = \sigma_1 = \cdots = \sigma_n$ and to try to find the best value for $\boldsymbol{\sigma}$. Indeed, when no a priori knowledge is available about the meaning of the attributes, the only choice is use spherical kernels (i. e., give same weight to each attribute). But one may expect that there is a better choice for shape of the kernel since many realworld databases contain attributes of very different natures. There may thus exist more appropriate scaling factors that give the right weight to the right features.

Usual methods for choosing parameters, based on exhaustive search become intractable as soon as the number of parameters exceeds two. Gradient descent algorithm is a common used method in nonlinear optimization when we know analytical expression of cost or error function with multiple parameters, which are independent variables of cost or error function. Here we propose a minimax approach: maximize the $w(\boldsymbol{\alpha})$ over the hyperplane coefficients and minimize an estimate of the generalization error over the set of kernel parameters based on gradient descent approach. This can be achieved by the following iterative procedure:

1) Initialize kernel parameters $\boldsymbol{\sigma} = [\sigma_1, \cdots, \sigma_n]^{\mathrm{T}}$;

2) Using a standard SVM algorithm, find the maximum of the quadratic form w, $\boldsymbol{\alpha}^0(\boldsymbol{\sigma}) = \arg\max_{\alpha} w(\boldsymbol{\alpha}, \boldsymbol{\sigma})$;

3) Update the parameter $\boldsymbol{\sigma}$ such that generalization error T is minimized using gradient descent method;

4) Go to step 2) or stop when the minimum of T is reached.

Solving step 3) requires estimating how T varies with

$\boldsymbol{\sigma}$. We will thus restrict ourselves to the case where kernel function can be differentiated with respect to $\boldsymbol{\sigma}$. Since $\boldsymbol{\alpha}^o$ depends implicitly on $\boldsymbol{\sigma}$, the total derivative of T^o with respect to $\boldsymbol{\sigma}$ is

$$\frac{\partial T^o}{\partial \sigma_k} = \frac{\partial T^o}{\partial \sigma_k}\Big|_{\boldsymbol{\alpha}^o \text{fixed}} + \frac{\partial T^o}{\partial \boldsymbol{\alpha}} \frac{\partial \boldsymbol{\alpha}}{\partial \sigma_k} \tag{8}$$

where $T^o = T(\boldsymbol{\alpha}^o, \boldsymbol{\sigma})$ and $k = 1, 2, \cdots, n$.

Having computed the gradient, a way of performing step 3) is to make a gradient step: $\boldsymbol{\sigma}(k+1) = \boldsymbol{\sigma}(k) - \eta \boldsymbol{H}_T^{-1} \frac{\partial T^o}{\partial \boldsymbol{\sigma}}$. Where η is positive learning factor, which is set small and eventually decreasing. $\boldsymbol{H}_T$ is Hessian matrix of T^o. It can also be calculated using DFP or BFGS method[8]. We will consider two ways of assessing the generalization error: validation error and estimate error.

2.2 Computing the gradient on validation error

Without loss of generality, we defined validation error as sum squared error on validation data set:

$$T = \frac{1}{2} \sum_{j=1}^{m} (f(X, Z_j, \boldsymbol{\alpha}, \boldsymbol{\sigma}) - y_{Z_j})^2 \tag{9}$$

where $f(\cdot)$ is function in (7), $(Z_j, y_{Z_j}), j = 1, \cdots, m$, is validation data set and X is training data set. Now we write out the formula of derivative of T with respect to $\boldsymbol{\sigma}$.

1) First part of (8).

$$\frac{\partial T}{\partial \sigma_k}\Big|_{\boldsymbol{\alpha} \text{ fixed}} = \sum_{j=1}^{m} (f(X, Z_j, \boldsymbol{\alpha}, \boldsymbol{\sigma}) - y_{Z_j}) \frac{\partial f(\cdot)}{\partial \sigma_k} \tag{10}$$

and

$$\frac{\partial f(\cdot)}{\partial \sigma_k} = \sum_{i=1}^{l} \alpha_i K(X_i, Z_j) \cdot \frac{(x_{ik} - z_{jk})^2}{\sigma_k^3} \tag{11}$$

where $k = 1, \cdots, n$.

2) Second part of (8).

$$\frac{\partial T}{\partial \boldsymbol{\alpha}} = \sum_{j=1}^{m} (f(X, Z_j, \boldsymbol{\alpha}, \boldsymbol{\sigma}) - y_{Z_j}) \frac{\partial f(\cdot)}{\partial \boldsymbol{\alpha}} \tag{12}$$

$$\frac{\partial f(\cdot)}{\partial \boldsymbol{\alpha}} = [k(X_1, Z_j), k(X_2, Z_j), \cdots, k(X_l, Z_j)] \tag{13}$$

If $\alpha_i = 0$, then ith item will be removed from the vector. If there are n support vectors, this column vector is $1 \times n$ vector.

For computing the derivative of $\boldsymbol{\alpha}$ with respect to $\boldsymbol{\sigma}$ of the kernel, we need an analytical formulation for $\boldsymbol{\alpha}$. First, we suppose that the points, which are not support vectors, are removed from train set. This assumption can be done without any loss of generalization since removing the point, which is not support vector, does not affect the solution. If there are n support vectors, then $\boldsymbol{H}\begin{bmatrix}\boldsymbol{\alpha}\\ b\end{bmatrix} = \begin{bmatrix}\boldsymbol{K}(X_i, X_j) & \boldsymbol{V}\\ \boldsymbol{V}^{\mathrm{T}} & 0\end{bmatrix}\begin{bmatrix}\boldsymbol{\alpha}\\ b\end{bmatrix} = \begin{bmatrix}\boldsymbol{Y}\\ 0\end{bmatrix}$, where $\boldsymbol{K}_{ij} = \boldsymbol{K}(X_i, X_j)$ is $n \times n$ matrix, $\boldsymbol{V}$ is $n \times 1$ vector which elements are all unity, $\boldsymbol{Y}$ is $n \times 1$ vector which contains support vectors' function values on training set. Then we have $(\boldsymbol{\alpha}, b) = \boldsymbol{H}^{-1}(Y, 0)^{\mathrm{T}}$.

$$\frac{\partial(\boldsymbol{\alpha}, b)}{\partial \sigma_k} = -\boldsymbol{H}^{-1} \frac{\partial \boldsymbol{H}}{\partial \sigma_k} \boldsymbol{H}^{-1} (\boldsymbol{\alpha}, b)^{\mathrm{T}} \tag{14}$$

and

$$\frac{\partial \boldsymbol{H}}{\partial \sigma_k} = \begin{pmatrix} \boldsymbol{K}(X_i, X_j) \dfrac{(x_{ik} - x_{jk})^2}{\sigma_k^3} & 0 \\ 0 & 0 \end{pmatrix} \tag{15}$$

2.3 Computing the gradient on error estimate

It was shown that there are also some error bounds for function regression for any probability distribution[7], which is stated below: consider performing regression with linear function $\mathfrak{L}$ on an inner product space X and fixed $\varepsilon \in \mathbf{R}^+$. There exits a constant c, such that for any probability distribution $\Im$ on $X \times \mathbf{R}$ with support in a ball of radius R around the origin, with probability $1 - \eta$ over l random examples S, the probability that a hypothesis $w \in \mathfrak{L}$ has output more than ε away from its true value is bounded by $\underset{\Im}{T}(f) \leqslant \frac{c}{l}\left(\frac{\|W\|_2^2 R^2 + S_{\mathrm{SE}}}{\varepsilon^2} \lg^2 l + \lg \frac{1}{\eta}\right)$ where S_{SE} is the sum squared error on the training set S. In particular we want to show how variations of the estimates relate to variations of the test error rather than how their values are related, so the item $\|w\|_2^2 R^2 + S_{\mathrm{SE}}$ is considered only when computing the derivative of T.

Derivative of $\|W\|_2^2 R^2$

$$\|W\|_2^2 = \langle W, W \rangle = \sum_{i,j=1}^{l} \alpha_i \alpha_j \boldsymbol{K}(X_i, X_j) \tag{16}$$

$$\frac{\partial \|W\|_2^2}{\partial \boldsymbol{\alpha}} = \left[2\sum_{i=1}^{l} \alpha_i \boldsymbol{K}(X_i, X_1), \cdots, 2\sum_{i=1}^{l} \alpha_i \boldsymbol{K}(X_i, X_l)\right] \tag{17}$$

If $\alpha_i = 0$, then ith item will be removed from the vector. If there are n support vectors, this column vector is $1 \times n$ vector.

$$\frac{\partial \|W\|_2^2}{\partial \sigma_k} = \sum_{i=1}^{l} \alpha_i \alpha_j \boldsymbol{K}(X_i, X_j) \frac{(x_{ik} - x_{jk})^2}{\sigma_k^3} \tag{18}$$

It was shown that the radius of the smallest sphere enclosing the train points can be achieved by solving the following quadratic problem[9]:

$$R^2 = \max_{\boldsymbol{\beta}} \sum_{i=1}^{l} \beta_i \boldsymbol{K}(X_i, X_j) - \sum_{i,j=1}^{l} \beta_i \beta_j \boldsymbol{K}(X_i, X_j) \tag{19}$$

subject to $\sum_{i=1}^{l} \beta_i = 1$ and $\beta_i \geqslant 0$ and

$$\frac{\partial R^2}{\partial \sigma_k} = \sum_{i=1}^{l} \beta_i^o \frac{\partial K(X_i, X_j)}{\partial \sigma_k} - \sum_{i,j=1}^{l} \beta_i^o \beta_j^o \frac{\partial K(X_i, X_j)}{\partial \sigma_k} \quad (20)$$

where $\boldsymbol{\beta}^o$ maximizes the previous quadratic form.

$$\frac{\partial(\|W\|_2^2 R^2)}{\partial \sigma_k}\Big|_{\alpha \text{ fixed}} = R^2 \frac{\partial(\|W\|_2^2)}{\partial \sigma_k} + \|W\|_2^2 \frac{\partial(R^2)}{\partial \sigma_k} \quad (21)$$

$$\frac{\partial(\|W\|_2^2 R^2)}{\partial \boldsymbol{\alpha}} = R^2 \frac{\partial \|W\|_2^2}{\partial \boldsymbol{\alpha}} \quad (22)$$

$$\frac{\partial(\|W\|_2^2 R^2)}{\partial \sigma_k} = \frac{\partial(\|W\|_2^2 R^2)}{\partial \sigma_k}\Big|_{\alpha \text{ fixed}} + R^2 \frac{\partial(\|W\|_2^2)}{\partial \boldsymbol{\alpha}} \frac{\partial \boldsymbol{\alpha}}{\partial \sigma_k} \quad (23)$$

Substitute (14), (21) and (22) into (23), we get derivative of $\|W\|_2^2 R^2$.

Derivative of S_{SE}

$$S_{SE} = \frac{1}{2} \sum_{j=1}^{l} (f(X, X_j, \boldsymbol{\alpha}, \boldsymbol{\sigma}) - y_j)^2 \quad (24)$$

The S_{SE} is defined on training set. The derivate of S_{SE} with respect to $\boldsymbol{\sigma}$ is the same as what we do in (2.2), provided that we replace test set $(Z_j, y_{Zj}), j = 1, \cdots, m$ with training set $(X_j, y_j), j = 1, \cdots, l$.

3 GA Optimization Approach

It is well known that gradient approach is very easily trapped by local minimum when it is used for multivariable optimization[10]. GA is a global gradient-free optimization, so the GA also be used in this paper for contrasting with gradient approach.

The GA is a stochastic global search method that mimics the metaphor of natural biological evolution. GAs operate on a population of potential solutions applying the principle of survival of the fittest to produce (hopefully) better and better approximations to a solution. At each generation, a new set of approximations is created by the process of selecting individuals according to their level of fitness in the problem domain and breeding them together using operators borrowed from natural genetics. This process leads to the evolution of populations of individuals that are better suited to the environment than the individuals that they were created from, just as in natural adaptation.

Individuals, or current approximations, are encoded as strings, chromosomes, composed of some alphabets, so that the genotypes (chromosome values) are uniquely mapped onto the decision variable domain. The most commonly used representation in GAs is the binary alphabet $\{0, 1\}$. The three major steps (operations) of GAs are reproduction, crossover, and mutation. Details of the process is introduced as follows.

1) Code the variable(s) $x_1, x_2, \cdots$, into a population of binary streams $(v_1, v_2, \cdots)$ of length m_i, with the population size prespecified as p_{size}.

2) Calculate the fitness value $e_{val}(v_i)$ for each chromosome v_i, $i = 1, \cdots, p_{size}$.

3) Find the total fitness of the population

$$F = \sum_i e_{val}(v_i) \quad (25)$$

4) Calculate the probability of a selection p_i for each chromosome v_i, $i = 1, \cdots, p_{size}$.

$$p_i = \frac{e_{val}(v_i)}{F} \quad (26)$$

5) Calculate the cumulative probability q_i for each chromosome v_i

$$q_i = \sum_{j=1}^{i} p_j \quad (27)$$

Then comes the reproduction process. Each time we select a single chromosome for a new population in the following way: generate a random number r from the range $[0, 1]$, if $r < q_1$, then select the first chromosome v_1; otherwise select the ith chromosome v_i, $2 \leqslant i \leqslant p_{size}$ such that $q_{i-1} < r \leqslant q_i$. In this way, some chromosomes whose p_i is relatively larger will be selected more than once. This is intuitively right since larger p_i means larger fitness value, and those chromosomes with smaller fitness value will gradually die off.

Next we apply the recombination operator, crossover, to the individuals in the new population. Here, an important parameter, probability of crossover, p_c is introduced. It gives us the expected number $p_c \times p_{size}$ of chromosomes, which undergo the crossover operation.

6) For each chromosome in the (new) population: generate a random number r from the range $[0, 1]$. If $r < p_c$, select given chromosome for crossover.

7) Mate the selected chromosomes randomly: for each pair of coupled chromosomes we generated a random integer number 'pos' from the range $[1, \cdots, m-1]$ (m is the number of binary bits in each chromosome). The number 'pos' indicates the position of the crossing point from where to the end of the bit stream the binary bits of the chromosome pair exchange.

8) Mutation. This operation is performed on a bit-by-bit basis. The parameter p_m, which is the probability of mutation, is introduced here. Every bit (in all chromosomes in the whole population) has an equal probability p_m of undergo mutation, i. e., change from 0 to 1 or vice versa. We proceed as follows: for each bit in the current population generate a random number r from the range $[0, 1]$, if $r < p_m$, mutate the bit.

Now, a new population is formed by the above three operations: reproduction, crossover, and mutation and is ready for the next evaluation. This iterative process repeats until some stop criteria are satisfied.

4 Experiments

4.1 Experiments setup

This section provides three examples to demonstrate the performance of our approach. We tried to select auto-matically the width σ_i of a RBF kernel,

$$k(X, Z) = \exp(-\sum_{i=1}^{n} \frac{(x_i - z_i)^2}{2\sigma_i^2}) \quad (28)$$

to get optimal solution for function regression.

The first experiment consists in finding the optimal value of parameter $\boldsymbol{\sigma}$ for approximating univariate function $sinc(x)$, which has single independent variable. The second experiment corresponds to optimize a single parameter $\boldsymbol{\sigma} = \sigma_1 = \sigma_2$ for a function with two independent variables. The third one is designed to optimize two parameters (σ_1, σ_2) for the same function used in the second experiment.

For time limits, we just designed some simple situations to test our approach. We sampled some points for the target function $f_T(X)$ evenly distributed over input interval of interest with function values contaminated by zero-mean Gaussian random noise to form training data set. The test data set comprises other points uniformly distributed over the same input interval with function values also contaminated by the same zero-mean Gaussian random noise, i. e., $y = f_T(X) + N(0, \gamma^2)$, where N is zero-mean Gaussian noise with standard deviation γ. We define average error over validation set as

$$A_{VE} = \text{sqrt}\{\sum_{j=1}^{m} (f(X, Z_j, \boldsymbol{\alpha}, \boldsymbol{\sigma}) - y_{Z_j})^2\} \quad (29)$$

and set up a criterion to detect over-fitting from statistical point of view: If $A_{VE} > \gamma$, it is under-fitting; If $A_{VE} = \gamma$, it is perfect approximation; If $A_{VE} < \gamma$, it is over-fitting. Because this SVM not only approximates the true function $f_T(X)$ but also takes noise into count.

We used the optimization toolbox of Matlab to perform quadratic programming and other gradient descent algorithm. It includes second order updates to improve the convergence speed. Because gradient descent approach is very easily trapped by local minimum, we repeated optimization procedure from different initial points and reserved the best one. For time limit, only gradient to validation error is considered.

Before proceeding experiments, we do some setup for GA algorithm. First we shall use binary codes for the parameters σ_i. We use 10 bits to represent the parameter, which is supposed to be in the range $[\sigma_{\min}, \sigma_{\max}]$, thus the relation between the value of the parameter and the corresponding binary string is $\sigma_i = \sigma_{\min} + \text{decimal}(\text{binary string}) \times (\sigma_{\max} - \sigma_{\min})/(2^{10} - 1)$.

We shall define the fitness value, since the larger the fitness value the better, and we want the approximation error to be minimized, thus we define the reciprocal of the error corresponding to each chromosome as the fitness value of that chromosome. This corresponds to our intuition that the smaller the approximation error, the better result we get, that means larger fitness value we get. Furthermore, we select the generation number = 20, the p_{size} = 20, i. e., the population has 20 binary strings within which we shall select the optimal one that maximize the fitness value.

4.2 Experiments results and analysis

1) The first experiment.

The target function is $sinc(x)$ with $x \in D = [-3, +3]$. The train set contains 31 points which evenly distributed over D and the test set has 169 points uniformly distributed over the same interval. The noise added is $N(0, 0.05^2)$. We set the kernel parameter $\boldsymbol{\sigma} \in [0.1, 5.0]$. The learning parameter C and ε belong to the set $\{1000, 10000, 100000\}$ and the set $\{0.01, 0.05, 0.1\}$ respectively. We use "GD" denote gradient descent method and "GA" for GA algorithm. The results are shown in table 1.

表 1　调整参数 $\boldsymbol{\sigma}$ 的结果

Table 1　Results by tuning parameter $\boldsymbol{\sigma}$

C		ε = 0.01	ε = 0.05	ε = 0.1
1000	GD	0.0514 (1.3938)	0.0538 (1.2439)	0.0602 (0.8518)
	GA	0.0604 (1.1470)	0.0614 (0.5709)	0.0643 (1.9078)
10000	GD	0.0518 (1.7404)	0.0528 (1.5533)	0.0602 (0.8518)
	GA	0.0588 (0.7509)	0.0610 (0.8064)	0.0613 (1.0756)
100000	GD	0.0529 (1.5442)	0.0514 (1.8885)	0.0602 (0.8518)
	GA	0.0554 (1.8045)	0.0563 (2.1579)	0.0598 (1.0453)

Note: In each entry in this table, the value in bracket is $\boldsymbol{\sigma}$ and the other value is average error A_{VE}.

Besides, we also tune the parameters C and $\boldsymbol{\sigma}$ simultaneously using GA algorithm. The results are shown in table 2.

表2 调整参数 C 和参数 σ 的结果

Table 2 Results by tuning parameters C and σ

ε	0.01	0.05	0.1
C	6047.3	8307.4	7548.3
σ	1.6045	0.7936	0.9053
A_{VE}	0.0473	0.0490	0.0523

2) The second experiment.

The target function is

$$f_T(X) = x_1^2 + x_2^2 - 5x_1x_2 \tag{30}$$

and $X \in D = [-2, +2] \times [-2, +2]$. The train set contains 81 points which evenly distributed over D and the test set has 119 points uniformly distributed over the same interval. The noise added is $N(0, 0.05^2)$. We set the kernel parameter $\sigma \in [0.1, 5.0]$. The learning parameter C and ε belong to the set $\{1000, 10000, 100000\}$ and the set $\{0.01, 0.05, 0.1\}$ respectively. The results are shown in table 3.

表3 第2次试验结果

Table 3 Results of the second experiment

C		ε		
		0.01	0.05	0.1
1000	GD	0.0532 (1.8623)	0.0529 (2.9545)	0.0594 (2.9748)
	GA	0.0941 (0.7392)	0.1313 (0.6903)	0.1739 (1.1530)
10000	GD	0.0509 (2.3637)	0.0503 (2.9613)	0.0591 (3.6360)
	GA	0.0861 (0.7256)	0.1209 (1.1904)	0.1650 (0.3127)
100000	GD	0.0545 (2.3727)	0.0507 (3.6615)	0.0646 (2.3529)
	GA	0.0843 (0.4093)	0.1299 (0.7764)	0.1570 (0.3927)

3) The third experiment.

The target function is (30) and $X \in D = [-2, +2] \times [-2, +2]$. The train set contains 81 points which evenly distributed over D and the test set has 119 points uniformly distributed over the same interval. The noise added is $N(0, 0.05^2)$. We set the kernel parameters $\sigma \in [0.1, 5.0] \times [0.1, 5.0]$. The learning parameter C and ε belong to the set $\{1000, 10000, 100000\}$ and the set $\{0.01, 0.05, 0.1\}$ respectively. The results are shown in table 4.

表4 第3次试验结果

Table 4 Results of the third expriment

C		ε		
		0.01	0.05	0.1
1000	GD	0.0529 (1.942,1.829)	0.0526 (2.992,2.915)	0.0546 (3.062,2.847)
	GA	0.0936 (1.310,0.407)	0.1295 (0.957,0.508)	0.1606 (1.905,0.235)
10000	GD	0.0503 (2.512,2.274)	0.0501 (3.140,2.809)	0.0519 (4.568,4.097)
	GA	0.0837 (1.996,0.890)	0.1206 (2.003,1.057)	0.1619 (0.591,0.106)
100000	GD	0.0540 (2.466,2.288)	0.0504 (3.091,2.845)	0.0574 (3.924,3.675)
	GA	0.0824 (1.297,1.068)	0.1278 (0.868,0.168)	0.1503 (1.035,1.133)

Remarks are as follows.

1) The values of A_{VE} come from three experiments are almost all bigger than 0.05. So the SVMs gotten from training are almost all under-fitting, except that two values from table 2.

2) The values of A_{VE} come from table 4 are smaller than those in table 3. Since the data pairs may have different natures. The results gotten from the situation in which there reserves a kernel parameter for every attribute will be better than those from the situation, which there is only one kernel parameter for all attributes.

3) The learning parameter C is a measure of trade-off between training error and generalization in table 2. If C is too big, the penalty on training error will be very large, then SVM tend to be overfit and validation error A_{VE} will be big. If C is too small, the penalty on training error will be very small, then generalization of SVM will be poor and A_{VE} will be also big. It is very clear from results listed in table 3 and 4. So there exits an optimal value for C such that training error and genera-lization are all best.

4) The parameter ε is a limit and error will be counted into training error during training only when the error is bigger than this limit. If this value is too big, i.e., the error between SVM and true function is bigger, then A_{VE} is big. If this value is too small, then SVM tend to be overfit and validation error A_{VE} is also big. It is verified with results in table 3 and 4.

5) In table 1, when C is bigger and ε is smaller, i.e., the penalty on training error is very large, then SVM tend to be overfit and validation error A_{VE} is big. If ε is too big, i.e., the error between S_{VM} and true function is bigger, then A_{VE} is big. That means that the results consist with the results in table 3 and 4.

6) The differences of kernel parameters between SVMs trained with different learning parameters are very big. This means that SVM is very sensitive to learning parameters, especially for ε. Because with change of ε, the number of support vectors and which point is support vector are altered correspondingly. The parameters of kernel also change dramatically. So SVMs will be not very robust.

7) The results gotten using GA algorithm is worse than that using gradient descent approach. The reason is that there has no enough time to run GA algorithm and there has no enough physical memory to generate more

population for each generation as well. The GA algorithm just guarantees that the result will be global optimization as long as generation goes to infinite.

5 Conclusion and Future Work

We have presented approach using gradient descent or GA algorithm for automatically tuning the kernel parameters of SVM to improve performance of the SVM. We tried to explain the results on artificial data set from theoretical point of view. It has shown that the results consist with theoretical analysis. This approach can also be used as feature selection. If σ_i is very large relative to other σ_j, then this feature can be omitted without harming the generalization.

In fact, all data set used in training process should be training data, because all data are used in training a SVM-quadratic programming for α_i and gradient descent or GA algorithm for σ_i. But the very interesting thing is that over-fitting effect remains low.

In term of future research, we try to use this approach on real world data set, and try to understand the phenomena as well.

References

[1] VAPNIK V. Estimation of dependences based on empirical Data[M]. New York:Springer Verlag, 1982.

[2] BARTLETT P L. For valid generalization, the size of the weights is more important than the size of the network [M]//MOZER M C, JORDAN M I, Petsche T. Advances in Neural Information Processing Systems. Cambridge, MA:The MIT Press, 1997:134-140.

[3] GEMAN S, BINENESTOCK E, DOURSAT R. Neural networks and the bias/variance dilemma[J]. Neural Computation, 1992, 4(1):1-58.

[4] VAPNIK V. Statistical learning theory [M]. New York: John Wiley and Sons, Inc, 1998.

[5] BURGES C J C. A tutorial on support vector machine for pattern recognition [J]. Data mining and knowledge discovery, 1998, 2(2):134-140.

[6] SMOLA A J, SCHÖLKOPF B. A tutorial on support vector regression[D]. NeuroCOLT2 technical report NC2-TR-1998-030. London:Royal Holloway College, 1998.

[7] CRISTIANINI N, SHAWE-TAYLOR J. An introduction to support vector machines and other kernel-based learning methods [M]. Cambridge: Cambridge University Press, 2000.

[8] KECMAN V. Learning and soft computing: Support vector machines, neural networks, and fuzzy logic models [M]. Cambridge, MA:The MIT Press, 2001.

[9] CHAPELLE O, VAPNIK V, BOUSQUET O, et al. Choosing multiple parameters for support vector machine [J]. Machine Learning, 2002, 46(1/2/3):131-159.

[10] GEN M, CHENG R. Genetic algorithms and engineering design[M]. New York:John Wiley & Sons, 1997.

(上接第 49 页)

[12] 王晓博,王国宏,阎红星,等. 利用位置和运动信息的目标识别[J]. 电光与控制,2008,15(10):5-9.

[13] PANNETIER B, BENAMEUR K, NIMIER V, et al. Ground moving target tracking with road constraint [C]//Proceeding of Signal Processing, Sensor Fusion, and Target Recognition XIII. Orlando:SPIE Press, 2004:138-149.

[14] SALMOND D, CLARK M, VINTER R, et al. Ground target modeling, tracking and prediction with road networks [C]//Proc. of 10th International Conference on Information Fusion. Quebec, Canada: Institution of Electronics and Electronic Engineering Computer Society Press, 2007:1-8.

[15] ADAM M F, JOHN L C, TARUNRAJ S, et al. Ground target tracking using terrain information [C]//Proceeding of 10th International Conference on Information Fusion. Quebec, Canada: Institution of Electronics and Electronic Engineering Computer Society Press, 2007:1-8.

[16] DEWANDARU A, SAID A M, MATORI A N. A novel map-matching algorithm to improve vehicle tracking system accuracy[C]//Proceeding of International Conference on Intelligent and Advanced Systems IEEE Press, 2007: 177-181.

[17] LIU F, ZHU S L, QI C H, et al. A novel adaptive map-matching algorithm in vehicular navigation system[C]// Proceeding of 7th International Conference on Fuzzy Systems and Knowledge Discovery. Yantai, China: IEEE Press, 2010:796-800.

[18] 华永平,刘砚一. 车载定位系统中综合地图匹配算法[J]. 现代雷达,2010,32(3):53-56.