International Journal of Machine Learning and Cybernetics (2019) 10:1529–1541
https://doi.org/10.1007/s13042-018-0832-7

ORIGINAL ARTICLE

Large-scale support vector regression with budgeted stochastic gradient descent

Zongxia Xie[1] · **Yingda Li**[1]

Received: 24 April 2017 / Accepted: 29 May 2018 / Published online: 7 June 2018

Abstract
Support vector regression (SVR) is a widely used regression technique for its competent performance. However, non-linear SVR is time consuming for large-scale tasks due to the dimension curse of kernelization. Recently, a budgeted stochastic gradient descent (BSGD) method has been developed to train large-scale kernelized SVC. In this paper, we extend the BSGD method to non-linear regression tasks. According to the performance of different budget maintenance strategies, we combine the stochastic gradient descent (SGD) method with the merging strategy. Experimental results on real-world datasets show that the proposed kernelized SVR with BSGD can achieve competent accuracy, with good computational efficiency compared to some state-of-the-art algorithms.

Keywords Support vector regression · Budget maintenance strategy · Stochastic gradient descent

1 Introduction

SVR is a widely used regression technique which is extended from support vector classification (SVC) by Boser et al. [1]. SVR has received much attention due to its competent performance in various fields, such as financial prediction, wind speed prediction, etc [2–4]. SVR is used to deal with small sample problems originally, and has obtained good results. However, with the massive increase of data quantities in last decades, how to train SVR models efficiently on large-scale datasets has become a hot topic.

Linear SVR has been studied firstly for big data. There have been many improved algorithms for large-scale linear SVR. In 2012, Chia-Hua Ho and Chih-Jen Lin extended state-of-the-art training methods for linear SVC to linear SVR [5]. There were mainly two types of methods for large-scale linear SVC. One was a Newton-type method for the primal-form [6]. The other was a coordinate descent approach for dual form [7]. Both of the two methods were extended to linear SVR in their paper. Similarly in 2015, Xie et al. [8] proposed a mini-batch quasi-Newton optimization algorithm to speed up the training process of linear SVR. The proposed method combined the first and second order gradient information estimated by a small set of training data with the the framework of the quasi-Newton algorithm. Very recently in 2016, Wang et al. [9] developed a novel e-Distance Weighted SVR (e-DWSVR) to address the limitations of original SVR and make it scalable to large-scale SVR through dual coordinate descent and averaged SGD strategies.

Although a lot of these algorithms focus on linear SVR problems, large-scale kernel SVR learning is less explored. The main reason is that training large-scale kernelized SVR faces the same challenge as kernelized SVC called the curse of kernelization [10]. There is a growing set of support vectors (SVs) in the iteration of optimization, and results in a non-linear growth in both model update time and prediction time with data size. Some algorithms have been proposed to solve this problem. In 2015, Zheng [11] proposed a smoothed objective function in the primal form to improve the computational efficiency combined with a conjugate gradient algorithm. But this paper didn't use the proposed method on large-scale data. Recently, Lu et al. [12] developed the kernel functional approximation techniques to solve large-scale kernel regression problems. The techniques including FOGD and NOGD were used to approximate a kernel function or kernel matrix, which transformed the kernel regression problem into an approximate linear regression problem.

✉ Zongxia Xie
caddiexie@hotmail.com

[1] School of Computer Software, Tianjin University, Tianjin, China

In this paper, we intend to use the famous budgeted SGD (BSGD) algorithm to solve large-scale non-linear SVR that was firstly proposed to solve large-scale kernelized SVC [10]. BSGD algorithm maintains a fixed number of SVs in the model, and updates them during the SGD training iterations. This algorithm is based on SGD, which is a recently popularized approach that can be efficient to deal with very large data or with data streams. Before using SGD, the objective of SVM is cast as an unconstrained optimization problem. Then this algorithm proceeds by iteratively receiving a labeled example and updating the model weights through SGD over the corresponding instantaneous objective function. Besides, the SGD method is combined with some budgeted maintenance strategies to control the number of SVs. Actually budgeted online SVM algorithms were proposed by Crammer et al. [13]. There were three main budget maintenance strategies: removal, projection and merging. In the case of removal, there were some different removal methods. In 2006, Cavallanti et al. [14] proposed a randomized budget perception (RBP) that removed a random SV. RBP achieved satisfactory performance though it was so simple. The method Forgetron was proposed to remove the oldest SV which was created when the quality of perception was the lowest [15], and its removal was considered to be the least hurtful to the model. Another removal strategy used by Wang et al. [10] was to remove the smallest SV with the smallest norm in order to achieve the goal of minimizing the averaged gradient error. Before projection, most previous work has focused on removing SVs. Orabona et al. [16] proposed a Projectron algorithm that projected the SV which would be removed in removal methods onto the other SVs in 2009. The projection was used to minimize the model weight degradation caused by removing a SV. For merging strategy, two SVs were merged into a newly created one which has the information of these two SVs, with the weights remaining unchanged [10]. It also has been proved to be the best budgeted strategy. Furthermore, Levesque et al. [17] proposed a method to combine removal and merging budgeted kernel SVMs trained with SGD in 2013, comprehensive empirical results have demonstrated the effectiveness of these algorithms. Therefore in this paper, we extend the BSGD algorithm for large-scale kernelized SVM to SVR and then the new algorithm can deal with large-scale kernelized SVR learning.

The rest of this paper is organized as follows. In Sect. 2, the formulation of SVR is introduced. In Sect. 3, we introduce the SGD method to linear and non-linear SVR. Then the BSGD method is introduced into kernel SVR for large-scale datasets. In Sect. 4, experiments on some UCI and LIBSVM datasets are conducted to evaluate the efficiency of our proposed method. Section 5 concludes the work.

2 Support vector regression (SVR)

Given a set of training samples $S = \{x_i, y_i\}$, $x_i \in R^n, y_i \in R, i = 1, 2, \ldots, l$, where instance $x_i \in R^n$ is a $n-$dimensional input vector, $y_i \in R$ are the corresponding target values and l is the number of samples. The goal of SVR is to find a function $f(x)$ that can obtain targets y_i for all the training data with high generalization performance [18]. As for the linear case, $f(x)$ is described as follows (we have omitted the bias term b because it hardly affects the experimental performance [5]),

$$f(x) = \langle w, x \rangle, \tag{1}$$

where $\langle \cdot, \cdot \rangle$ denotes the dot product. In order to obtain the function f, SVR solves the following regularized optimization problems:

$$\min_w \ P(w) = \frac{1}{2} w^T w + \frac{C}{l} \sum_{i=1}^{l} L(w, x_i, y_i), \tag{2}$$

where $L(w, x_i, y_i) = \max(|w^T x_i - y_i| - \varepsilon, 0)$ is the epsilon-insensitive loss function, the parameter C is the regularization parameter and the parameter ε is given so that the loss is zero if $|w^T x_i - y_i| \leq \varepsilon$.

Furtherly, kernel functions are used in SVR to solve non-linear problems [19]. The input data are mapped into a high dimensional feature space through a nonlinear function Φ and then x is replaced with $\Phi(x)$, the optimization problem becomes as follows:

$$\min_w \ P(w) = \frac{1}{2} w^T w + \frac{C}{l} \sum_{i=1}^{l} L(w, \Phi(x_i), y_i). \tag{3}$$

In order to solve this problem, we can use the dual problem of SVR interchangeably by employing the lagrange multipliers method [5], then we can get

$$\begin{aligned} w &= \sum_{j=1}^{l} (\alpha_j - \alpha_j^*) \Phi(x_j) \\ &= \sum_{j=1}^{l} \theta_j \Phi(x_j), \end{aligned} \tag{4}$$

where θ is defined as the difference between lagrange multipliers α and α^*. Then the objective function $f(x)$ becomes

$$f(x) = \langle w, \Phi(x) \rangle = \sum_{i=1}^{l} \theta_i \langle \Phi(x_i), \Phi(x) \rangle = \sum_{i=1}^{l} \theta_i K(x_i, x).$$

After employing Lagrange multipliers method, we only need to compute $\langle \Phi(x_i), \Phi(x) \rangle$ to get $f(x)$. Gaussian kernel is one of the most used kernels. Its formula is as follows:

$$K(x, x') = e^{-\frac{||x-x'||^2}{2\sigma^2}}.$$

3 Budgeted stochastic gradient descent for SVR

In this section, we introduce the classical SGD algorithms to large-scale SVR firstly. Then the budget maintenance strategies are introduced to combine with SGD for solving large-scale kernelized SVR.

3.1 Stochastic gradient descent for SVR

SGD is an appealing algorithm which is widely used in the process of learning SVMs [20–23]. With SGD, the gradient is approximated by evaluating on training samples, over which the optimal model weight can be achieved with a small number of iterations. Therefore the SVM models with the SGD algorithm are efficient. We introduce the SGD algorithm to SVR and show its details in the following.

SGD works iteratively. It starts with an initial value of the model weight w_1, and at $t-$th round it updates the current weight w_t as

$$w_{t+1} \leftarrow w_t - \eta_t \nabla_w P_t(w_t), \tag{5}$$

where η_t is a learning rate. Different SGD algorithms have different η_t. Pegasos and Norma are the main two SGD algorithms [22, 23]. η_t of the former is C / t, while the latter is $\eta/\sqrt{t}$ (η is a constant). Here we choose Pegasos because it is a improved SGD method [23]. $\nabla_w P_t(w_t)$ is the sub-gradient of the objective function $P_t(w)$, that is defined on the latest example (x_t, y_t). That is,

$$P_t(w) = \frac{1}{2} w^T w + C * L(w, x_t, y_t). \tag{6}$$

Therefore Eq. 5 can be written as

$$w_{t+1} \leftarrow (1 - \eta_t) w_t + \beta_t x_t, \tag{7}$$

where

$$\beta_t \leftarrow \begin{cases} 0 & if\ |w^T x_t - y_t| \le \varepsilon \\ -C & if\ w^T x_t - y_t > \varepsilon \\ C & if\ w^T x_t - y_t < -\varepsilon\ . \end{cases} \tag{8}$$

From Eq. 8, we can know that if $|w^T x_i - y_i| \le \varepsilon$, $\beta_t = 0$. Then the sample (x_t, y_t) is useless to the output and can therefore be ignored.

3.2 Kernelized SGD for SVR

The SGD algorithm can be easily kernelized combined with kernels to train nonlinear SVR models. When introducing a non-linear function Φ that maps x from the input to the feature space and replacing x with $\Phi(x)$, w_t can be described as

$$w_t = \sum_{j=1}^{t} \theta_j \Phi(x_j),$$

where

$$\theta_j = \beta_j \prod_{k=j+1}^{t} (1 - \eta_k).$$

After kernelization, the Expression 7 of the weight w is converted to the following:

$$w_{t+1} \leftarrow (1 - \eta_t) w_t + \beta_t \Phi(x_t). \tag{9}$$

According to above equation of θ, the sample can be ignored if the value of θ is zero. Therefore the optimization problem of kernel SVR can be described by either a weight vector w or a set of θ and samples. We use pseudo-code to show the process of training the non-linear SVR in Algorithm 1. In the algorithm, Gaussian kernel is used. We get the value of β in different cases (line 8 to 13 of Algorithm 1). Then we update the value of θ continually, finally output the optimal function.

Algorithm 1 SGD for kernel SVR.

1:Input: samples$S = \{x_i, y_i\}$, $x_i \in R^n, y_i \in R, i = 1, 2, \cdots, l$, kernel k, penalty factor C, learning rate η, cycle index $Epoch$, SVs set I_b, #SVs b.
2:Initialize: $b = 0, I_b = \{\}$.
3:for $epochi = 1, \cdots, Epoch$ do
4: for i=1,2,$\cdots$,l do
5: $t = (epochi - 1) * l + i$; l is the number of samples.
6: receive $(x_t, y_t) = (x_m, y_m)$, m is randomly determined.
7: if $t == 1$
8: $b = b + 1$, $\theta_1 = \beta_1 = 1$, $I_b = I_b \bigcup x_t$;
9: else
10: if $\sum k(x_t, I_b)\theta_j - y_t > \varepsilon$
11: $b = b + 1, \beta_b = C, I_b = I_b \bigcup x_t$,
12: update $\theta = \{\theta_1, \theta_2, \cdots, \theta_b\}$, where $\theta_j = \beta_j \prod_{k=j+1}^{b} (1 - \eta_k)$.
13: end if
14: if $\sum k(x_t, I_b)\theta_j - y_t < -\varepsilon$
15: $b = b + 1, \beta_b = -C, I_b = I_b \bigcup x_t$,
16: update $\theta = \{\theta_1, \theta_2, \cdots, \theta_b\}$, where $\theta_j = \beta_j \prod_{k=j+1}^{b} (1 - \eta_k)$.
17: end if
18: end if
19: end for
20:end for
21:Output: $f_{t+1}(x) = \sum_{i=1}^{b} k(I_b, x)\theta_i$. $I_b = \{SV_1, SV_2, \ldots, SV_b\}$.

3.3 Budgeted maintenance strategies

In this subsection, we employ a budget maintenance strategy to solve non-linear SVR models. The strategy predefines a fixed budget B, which means that the number of SVs can't exceed B during iterations. When the number of SVs exceeds B, the strategy will execute a budget maintenance step. Generally there are three main budget maintenance strategies: merging, projection, and removal. According to the performance of kinds of strategies in the paper [10], we choose to combine SGD method with a merging strategy. Then the process of employing the merging strategy to SVR is described.

The goal of budgeted strategy is to minimize the averaged gradient error represented by $\bar{E}$ as paper [10]. The goal is solved by minimizing the current gradient error $\|E_t\|$ at each round according to the theorem defined in the original paper [10]. Define the gradient error $E_t = \frac{\triangle_t}{\eta_t}$ and η_t is the learning rate.

Therefore we finally solve the following objective function:

$$min \| \triangle_t \|^2, \tag{10}$$

where $\| \triangle_t \|$ is the weight degradation. In the following, we address Eq. 10 using the merging strategy which can merge two SVs to a newly created one. First, for the current weight, we suppose to merge $\Phi(x_m)$ $\Phi(x_n)$ to a new one M, then M is represented by

$$M = \frac{\alpha_m \Phi(x_m) + \alpha_n \Phi(x_n)}{(\alpha_m + \alpha_n)},$$

assuming that $\alpha_m + \alpha_n \neq 0$. In order to maintain the weight unchanged, the coefficient of M is set to $\alpha_m + \alpha_n$. In order to get M, this problem is turned into another one which find an input vector z whose image $\Phi(z)$ is at the minimum distance from the $M's$ [10]. Then the objective function is converted to the following:

$$\begin{aligned} &\min_z \ \|M - \Phi(z)\|^2 \\ &= \min_z \ \left(M^T M + \Phi(z)^T \Phi(z) - 2M^T \Phi(z)\right). \end{aligned} \tag{11}$$

In this paper, the Gaussian kernel is used in the experiments. It is a radial kernel, so the kernel can be expressed as $k(x, x') = \tilde{k}(\|x - x'\|^2)$. $k(x, x') \leq 1$, $k(x, x) = 1$. With this property, Eq. 11 can be written as:

$$\min_z \ \left(2 - 2M^T \Phi(z)\right). \tag{12}$$

Then the Eq. 12 can be reduced to:

$$\begin{aligned}\max_z f(z) &= \max_z M^T\Phi(z)\\ &= (\tfrac{\alpha_m\Phi(x_m)+\alpha_n\Phi(x_n)}{(\alpha_m+\alpha_n)})^T\Phi(z)\\ &= \tfrac{1}{\alpha_m+\alpha_n}((\alpha_m\Phi(x_m)^T\Phi(z)+(\alpha_n\Phi(x_n)^T\Phi(z))\\ &= \tfrac{1}{\alpha_m+\alpha_n}(\alpha_m\tilde{k}(\|x_m-z\|^2)+\alpha_n\tilde{k}(\|x_n-z\|^2)),\end{aligned} \tag{13}$$

Next we take the directive of 13 with respect to z,

$$\nabla_z f(z) = 0.$$

Then the formula of z simplified by h is obtained:

$$z = hx_m + (1-h)x_n, \tag{14}$$

where $h = \frac{\alpha_m\tilde{k}'(\|x_m-z\|^2)}{\alpha_m\tilde{k}'(\|x_m-z\|^2)+\alpha_n\tilde{k}'(\|x_n-z\|^2)}$, and $\tilde{k}'(x)$ is the first derivative of $\tilde{k}$. After expressing z, we take Eq. 14 into Eq. 13, then the objective function can be simplified to find the optimal h:

$$\max_h \left(\frac{\alpha_m}{\alpha_m+\alpha_n}k_{1-h}(x_m,x_n)+\frac{\alpha_n}{\alpha_m+\alpha_n}k_h(x_m,x_n)\right),$$

where $k_h(x,x') = k(hx,hx')$. Now we have transformed the objective function into the search for the optimal h. Learning from the methods of [10], we use the golden search method to find it. Then the optimal z can be obtained from Eq. 14.

After getting the optimal z, the original objective function to be solved as follows:

$$\| \triangle_t \|^2 \equiv \min \|\alpha_m\Phi(x_m)+\alpha_n\Phi(x_n)-\alpha_z\Phi(x_z)\|^2. \tag{15}$$

There is still a problem how to choose what pair of SVs to merge. To reduce the computational cost, we firstly compute the smallest value of $\|\alpha_m\|^2$ as one of pair, then we choose another one through a loop of SVs. After finishing these series of work, the algorithm's framework of BSGD is listed in Algorithm 2. At the beginning, a budget B is predefined to bound the number of SVs and a parameter b is defined to record the number of current SVs. In the iteration, when b exceeds B, the budgeted strategy is executed.

Algorithm 2 BSGD for kernel SVR.

1:Input: samples$S = \{x_i, y_i\}$, $x_i \in R^n, y_i \in R, i = 1,2,\cdots,l$, kernel k, penalty factor C, learning rate η, cycle index $Epoch$, SVs set I_b, #SVs b, Budget B.
2:Initialize: $b = 0, I_b = \{\}$.
3:for $epochi = 1,\cdots,Epoch$ do
4: for i=1,2,$\cdots$,l do
5: $t = (epochi-1)*l+i$; l is the number of samples.
6: receive $(x_t, y_t) = (x_m, y_m)$, m is randomly determined.
7: if $t == 1$
8: $b = b+1$, $\theta_1 = \beta_1 = 1$, $I_b = I_b \bigcup x_t$;
9: else
10: if $\sum k(x_t, I_b)\theta_j - y_t > \varepsilon$
11: $b = b+1, \beta_b = C, I_b = I_b \bigcup x_t$,
12: update $\theta = \{\theta_1, \theta_2, \cdots, \theta_b\}$, where $\theta_j = \beta_j \prod_{k=j+1}^{b}(1-\eta_k)$.
13: end if
14: if $\sum k(x_t, I_b)\theta_j - y_t < -\varepsilon$
15: $b = b+1, \beta_b = -C, I_b = I_b \bigcup x_t$,
16: update $\theta = \{\theta_1, \theta_2, \cdots, \theta_b\}$, where $\theta_j = \beta_j \prod_{k=j+1}^{b}(1-\eta_k)$.
17: end if
18: if $b > B$
19: $w_{t+1} \leftarrow w_{t+1} - \Delta_t$;//$\Delta_t$ is expressed as Equation 15.
20: $b = b-1$;
21: end if
22: end if
23: end for
24:end for
25:Output:$f_{t+1}(x) = \sum_{i=1}^{b} k(I_b, x)\theta_i$. $I_b = \{SV_1, SV_2, \ldots, SV_b\}$.

4 Experiments

In this section, we evaluate the BSGD method with a merging strategy and compare it to related algorithms including original SGD method [23] and LIBSVM [24] on 9 datasets.

4.1 Experimental settings

We select total 9 datasets in our experiments, and all expect 3 Wind-speed datasets are publicly available at LIBSVM or UCI datasets. We preprocess the datasets as follows. Firstly, all features and target values of datasets are normalized into [0, 1] excepting 3 Wind-speed datasets. Secondly, in order to ensure the experiments for large-scale datasets, datasets Cadata and CASP are repeated for 10 times. Cpusmall and Space_ga are repeated for 100 times. The descriptions of these datasets including the numbers of features, training and test data are listed in Table 1.

Besides, the selection of optimal parameters is accomplished through a grid search conducted on each parameter. For the non-linear SVR, RBF kernel is used in the algorithms. SVR with the SGD or BSGD method has a cross loop of regularization $C = [10^{-4}, 10^{-3}, \ldots, 10^{4}]$, RBF kernel width $\sigma = [10^{-4}, 10^{-3}, \ldots, 10^{4}]$ and $\epsilon = [2^{-10}, 2^{-9}, \ldots, 2^{0}]$. While LIBSVM has a cross loop of $penalty\_C = [2^{-2}, 2^{-1}, \ldots, 2^{6}]$, $\epsilon = [2^{-10}, 2^{-9}, \ldots, 2^{2}]$ and RBF kernel width $\sigma = [10^{-4}, 10^{-3}, \ldots, 10^{4}]$. Table 2 lists the optimal parameter values.

All experiments are conducted on a 8G RAM 3.6 GHz Intel core. Our proposed algorithms are implemented in MATLAB 2013B.

Table 1 Properties of different datasets

Data	#Instances	Training size	Testing size	#Features
Space_ga	3107	250,000	607	6
Cpusmall	8192	600,000	2192	12
TFIDF-2006	16,087	16,087	3308	150,360
Cadata	20,640	180,000	2640	8
CASP	45,730	400,000	5730	9
Wind-ningxia	51,840	450,000	6840	24
Wind-farm	246,360	246,360	26,360	6
MSD	515,345	463,715	51,630	90
Wind-saihanba	521,280	521,280	60,000	12

Note that these datasets are sorted according to the number of instances that varies from 3107 to 521,280

4.2 Experimental results and analysis

In this subsection, the performance of these algorithms on large-scale datasets described above are listed. Mean squared error (MSE), R-square, training time, test time and the number of SVs are 5 indexes of evaluating the proposed algorithms. Tables 3, 4 and 5 details the performance of these algorithms on different datasets. In the tables, numbers in brackets denote the number of SVs and the test time created by evaluating each algorithm. The best MSE, R-square, the shortest training time and test time are indicated in bold.

Then Figs. 1, 2, 3, 4, 5, 6, 7, 8 and 9 show the accuracy, training time, test time and the number of SVs of each algorithms on all datasets change with the increase of sample numbers. The algorithms are represented by the line with different shapes and colors. From Figs. 1, 2, 3, 4, 5, 6, 7, 8 and 9, we know that if the size of dataset is small, MSE is high, and the training time and test time of the BSGD methods are not significantly shorter than those of libsvm. This shows that the BSGD methods are more suitable for large datasets compared with libsvm.

As is shown in tables and figures, obviously the BSGD algorithm has achieved competent accuracy with significantly improved computational efficiency. Pegasos and LIBSVM have an advantage in accuracy owing to their large number of SVs in SVM modeling, thus resulting a high computational cost. For the two budget methods, we sacrifice a little accuracy to achieve a great improvement in efficiency.

Table 2 The optimal parameters of different algorithms

Datasets	SVR_SGD/BSGD(RBF)			LIBSVM(RBF)		
	ϵ	C	γ	ϵ	C	γ
Space_ga	2^{-3}	10^{1}	10^{-1}	2^{-3}	2^{1}	10^{-2}
Cpusmall	2^{-4}	10^{2}	10^{-1}	2^{-4}	2^{1}	10^{-1}
TFIDF-2006	2^{-1}	10^{1}	10^{0}	2^{-6}	2^{6}	10^{0}
Cadata	2^{-4}	10^{3}	10^{-1}	2^{-4}	2^{1}	10^{0}
CASP	2^{-3}	10^{3}	10^{-2}	2^{-3}	2^{6}	10^{-1}
Wind-ningxia	2^{-1}	10^{-2}	10^{-4}	2^{-5}	2^{3}	10^{-4}
Wind-farm	2^{-1}	10^{3}	10^{2}	2^{-3}	2^{3}	10^{-3}
MSD	2^{-4}	10^{1}	10^{-1}	2^{-4}	2^{5}	10^{-1}
Wind-saihanba	2^{-6}	10^{2}	10^{2}	2^{-4}	2^{3}	10^{-4}

Table 3 Testing MSE and the number of SVs using the different algorithms

Datasets	SVR_SGD (Pegasos) MSE (#SVs)	SVR_BSGD B = 500	SVR_BSGD B = 200	LIBSVM
Space_ga	0.0023 (4749)	0.0023 (500)	0.0024 (200)	**0.0018** (3685)
Cpusmall	**0.0022** (5938)	**0.0022** (500)	0.0024 (200)	0.0023 (4314)
TFIDF-2006	**0.1403** (1878)	0.1532 (500)	0.1540 (200)	**0.1403** (11,410)
Cadata	**0.0138** (61,692)	0.0140 (500)	0.0142 (200)	0.0140 (51871)
CASP	0.0432 (218,649)	0.0452 (500)	0.0486 (200)	**0.0424** (206,490)
Wind-ningxia	0.4131 (127,467)	0.4883 (500)	0.5289 (200)	**0.3036** (353,426)
Wind-farm	4.2212 (183,413)	4.2485 (500)	4.2641 (200)	**3.9364** (232,965)
MSD	0.0153 (924)	0.0153 (500)	0.0153 (200)	**0.0111** (111,601)
Wind-saihanba	0.8557 (256,773)	0.8606 (500)	0.8626 (200)	**0.8352** (413,745)

B number of budgeted support vectors, *BSGD* budgeted SGD, *LIBSVM* classical support vector regression method, *SGD* stochastic gradient descent, *SVR* support vector regression

The best MSE, R-square, the shortest training time and test time are indicated in bold

Table 4 Training time (s) and test time (s) using the different algorithms

Datasets	SVR_SGD (Pegasos) Trainging time (testing time)	SVR_BSGD B = 500	SVR_BSGD B = 200	LIBSVM
Space_ga	24.88 (0.0821)	17.31 (0.0078)	**11.03 (0.0017)**	33.28 (0.0431)
Cpusmall	109.11 (0.2917)	43.58 (0.0252)	**27.90 (0.0098)**	334.76 (0.3034)
TFIDF-2006	1658.66 (5.8231)	1411.99 (2.5539)	**1015.13 (1.0586)**	1611.85 (302.4381)
Cadata	417.11 (2.9933)	146.15 (0.0254)	**66.97 (0.0102)**	817.01 (3.0857)
CASP	4735.20 (30.7338)	547.14 (0.0528)	**304.88 (0.0274)**	21121.77 (37.1780)
Wind-ningxia	7539.24 (16.3556)	342.73 (0.0660)	**168.00 (0.0239)**	12968.19 (122.7013)
Wind-farm	1722.33 (61.1324)	540.45 (0.2331)	**218.78 (0.0723)**	2452.69 (73.4664)
MSD	62.48 (0.9606)	56.90 (0.6189)	**31.32 (0.2279)**	15294.16 (751.3881)
Wind-saihanba	8421.00 (300.0160)	696.58 (0.5143)	**340.07 (0.1834)**	19608.94 (932.5680)

B number of budgeted support vectors, *BSGD* budgeted SGD, *LIBSVM* classical support vector regression method, *SGD* stochastic gradient descent, *SVR* support vector regression

The best MSE, R-square, the shortest training time and test time are indicated in bold

Table 5 R-square of different algorithms

Datasets	SVR_SGD (Pegasos) R-square	SVR_ BSGD B = 500	SVR_ BSGD B = 200	LIBSVM
Space_ga	0.3723	0.3723	0.3451	**0.5088**
Cpusmall	**0.9325**	**0.9325**	0.9263	0.9294
TFIDF-2006	**0.5147**	0.4701	0.4673	**0.5147**
Cadata	**0.7780**	0.7748	0.7715	0.7748
CASP	0.4931	0.4697	0.4298	**0.5025**
Wind-ningxia	0.9572	0.9495	0.9452	**0.9686**
Wind-farm	0.8547	0.8538	0.8532	**0.8645**
MSD	0.0377	0.0377	0.0377	**0.3091**
Wind-saihanba	0.9472	0.9469	0.9468	**0.9485**

B number of budgeted support vectors, *BSGD* budgeted SGD, *LIBSVM* classical support vector regression method, *SGD* stochastic gradient descent, *SVR* support vector regression

The best MSE, R-square, the shortest training time and test time are indicated in bold

Table 6 Complexities on different algorithms

Algorithms	Update time	Space
BSGD + merging	$O(B)$	$O(B^2)$
SVR_SGD	$O(\#SVs)$	$O(l^2)$

In the table, B is the pre-defined budget equal to the number of SVs, and #*SVs* is the number of SVs in models, l is the number of samples. As reference [10], update time listing in the table includes both model update time and budget maintenance time and space corresponds to space needed to store the model and perform model update and budget maintenance

For the training time, the proposed BSGD method is multiple times even dozens of times faster than non-budget algorithms. For the test time, the proposed BSGD method is dozens of times even hundreds of times faster than others. Comparing with the traditional methods in which the number of SVs increases dramatically, we use a budget method to control the number of SVs at a fixed value. Besides, with

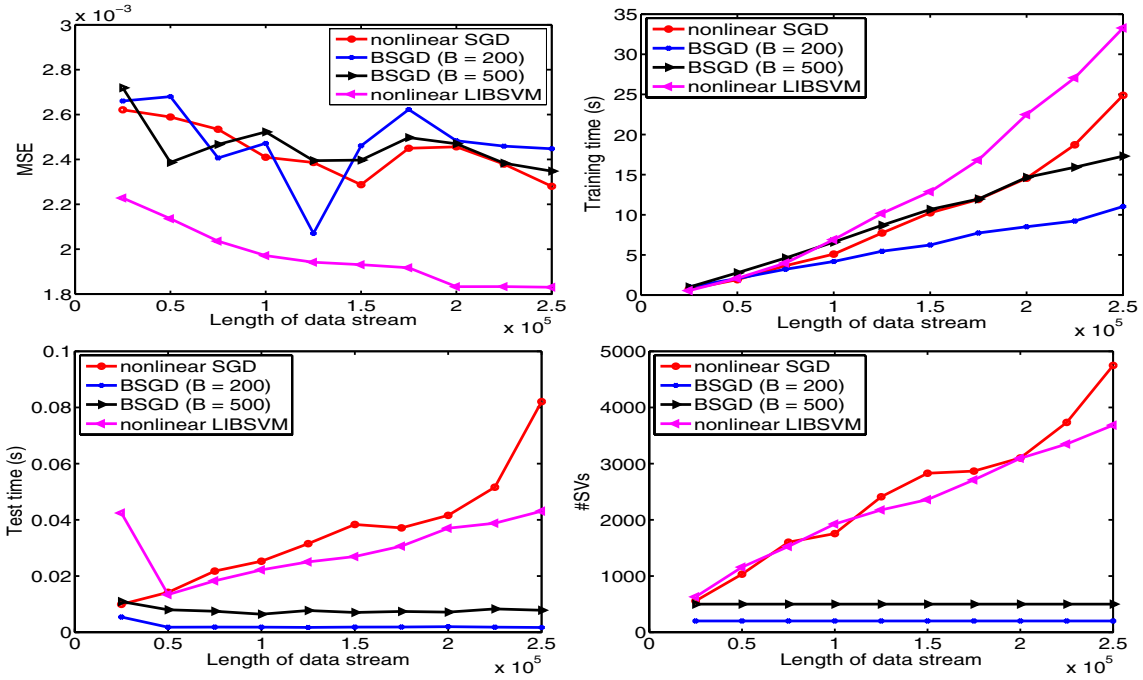

Fig. 1 MSE, training time, test time and SVs of Space_ga

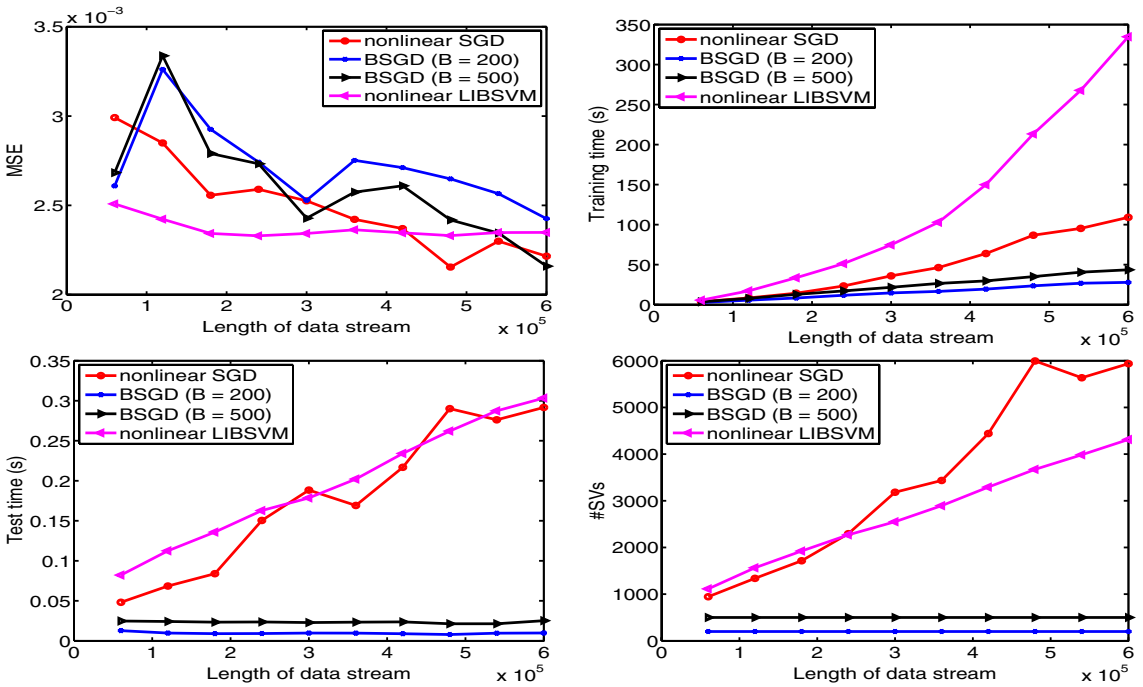

Fig. 2 MSE, training time, test time and SVs of Cpusmall

the increase of training sample numbers, the MSE has a tendency to decrease. Of course the training time and test time will increase gradually. However, the training time and test time of the BSGD methods increase slower than non-budget methods. Especially for test time, it increases slowly, even there is almost no increase in the BSGD methods. This proves the efficiency and effectiveness of our proposed BSGD method.

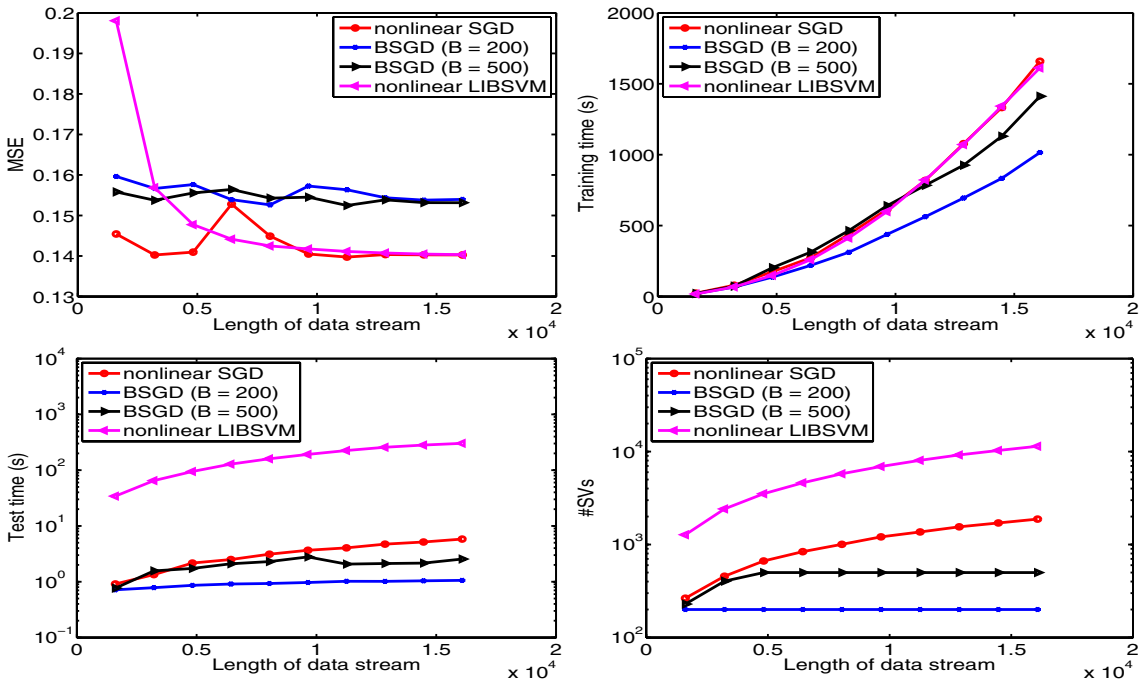

Fig. 3 MSE, training time, test time and SVs of TFIDF-2006

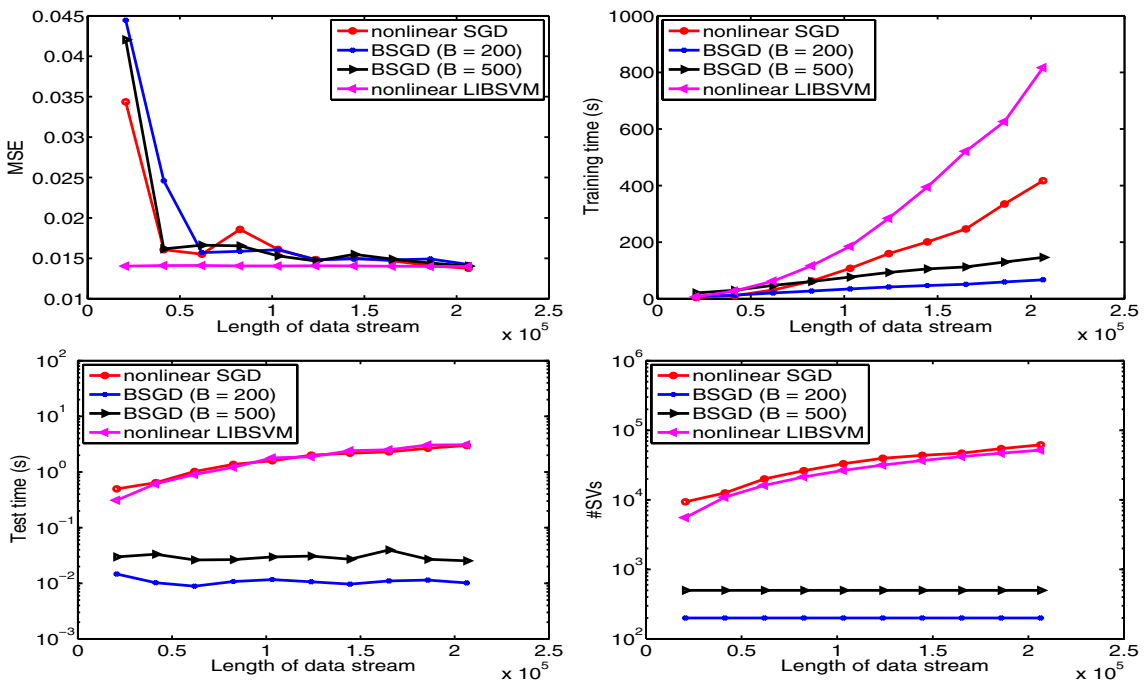

Fig. 4 MSE, training time, test time and SVs of Cadata

Finally, we present the MSE and training time of SVR with BSGD method on CASP dataset with a increase of budget size in Fig. 10. It shows that with the increase of budget size, the MSEs have the tendency to decrease, and the training times increase gradually with the increase of budget sizes. Tables 3 and 4 also show the results of the BSGD algorithm with two budgets including 200 and 500. With the decrease of budget size, the accuracy and training

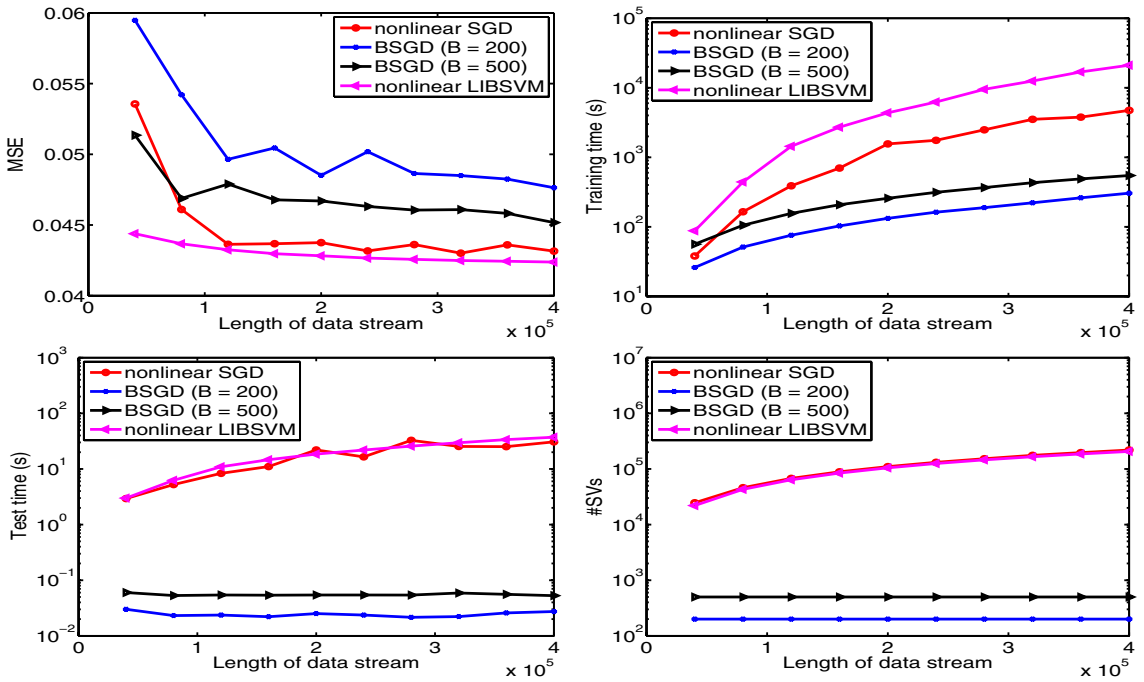

Fig. 5 MSE, training time, test time and SVs of CASP

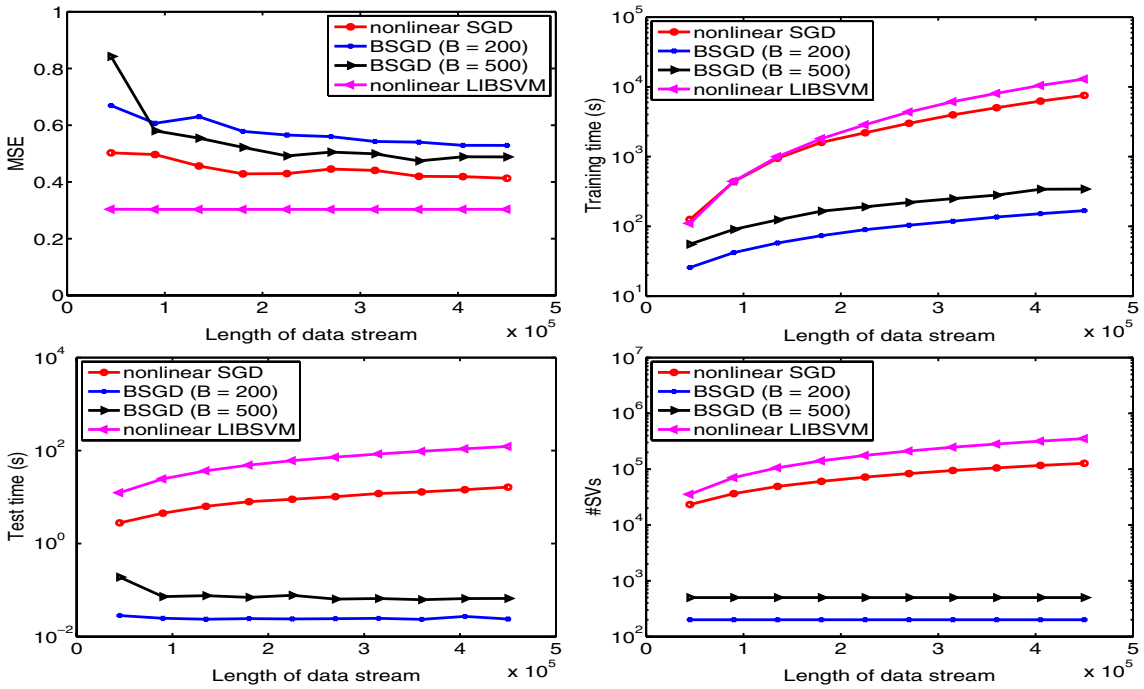

Fig. 6 MSE, training time, test time and SVs of Wind-ningxia

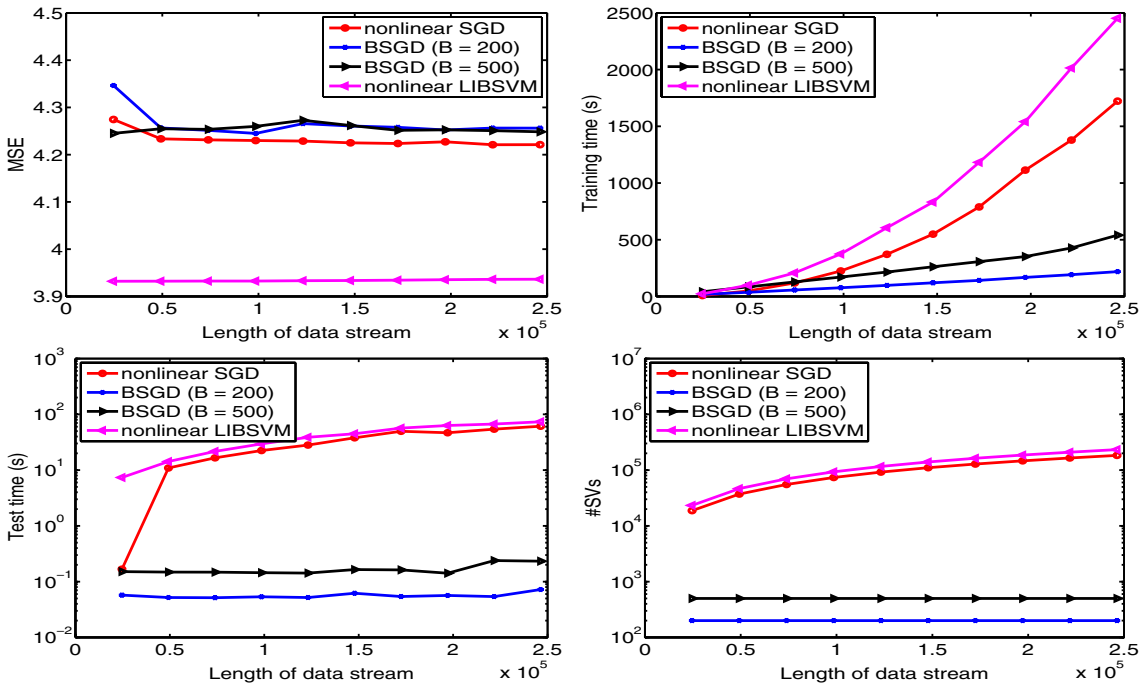

Fig. 7 MSE, training time, test time and SVs of Wind-speed

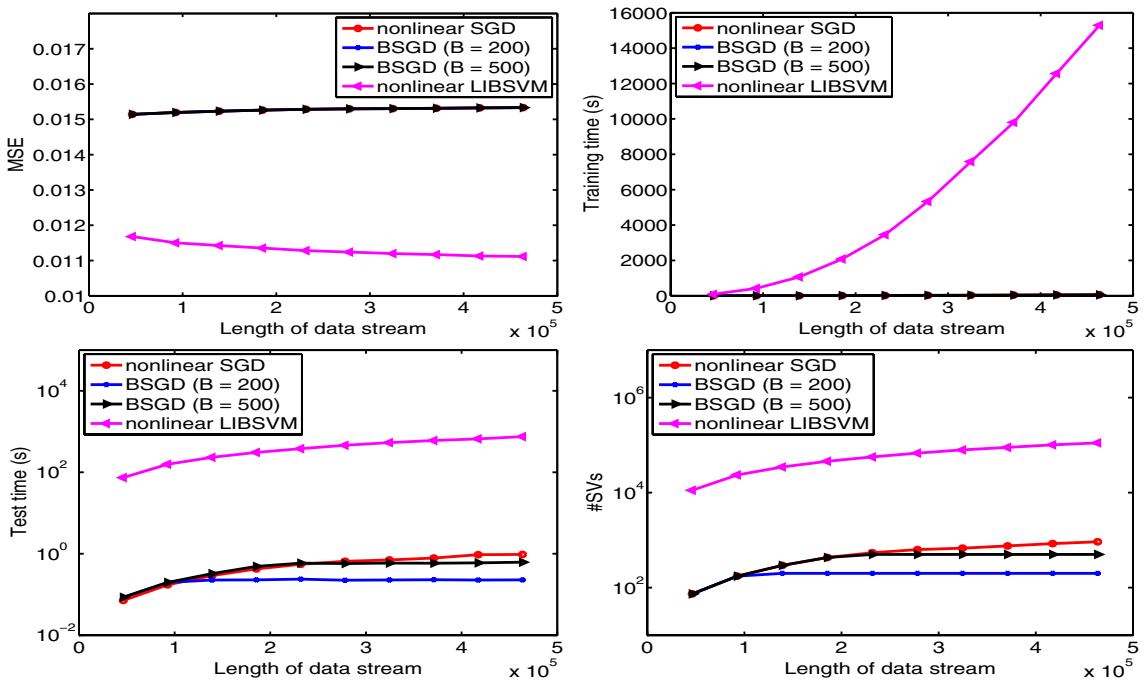

Fig. 8 MSE, training time, test time and SVs of MSD

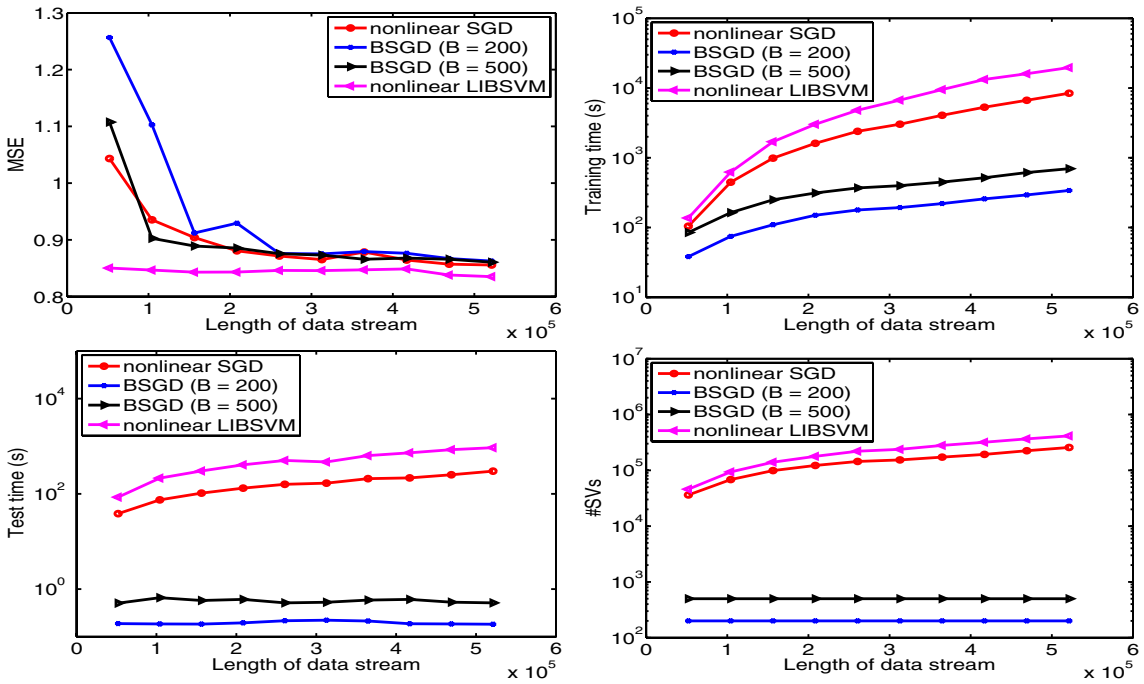

Fig. 9 MSE, training time, test time and SVs of Wind-saihanba

time decrease gradually. So we can balance accuracy and efficiency through regulating the size of budget. As for how to choose suitable budget value, normally we can choose the rough budget value on when the precision increases relatively slow. However for different datasets, the budget value can be choosed depending on the preference on efficiency or accuracy.

Table 6 shows the update time and space of different methods. We can know that our method has constant space and constant time complexity per update, which is $O(B)$. However, time and space complexity of SGD method is growing with the number of SVs and samples, respectively. Therefore, for large datasets, the complexity of SGD will become large while our method is only related the predefined number B.

4.3 Experimental summary

In this subsection, we summarize the conclusions obtained in our experiments.

1. Our proposed non-linear SVR with the BSGD method can solve the large-scale non-linear regression problems effectively, and avoid the huge time cost created by the traditional methods.
2. Through regulating the size of budget, we can balance accuracy and efficiency of the SVR models when dealing with non-linear regression problems.

5 Conclusions

In this paper, we propose a BSGD method to slove the large-scale non-linear SVR problems. It extends the SGD method to the optimization process of SVR, and combine with a budget method to control the number of SVs to overcome the curse of kernelization. Our empirical results on different large-scale datases demonstrate that the proposed SVR with the BSGD method can solve the large-scale non-linear regression problems well. It can achieve competent accuracy with significantly improved computational efficiency. Besides, through controlling the size of budget, we can balance the accuracy and efficiency of non-linear regression tasks.

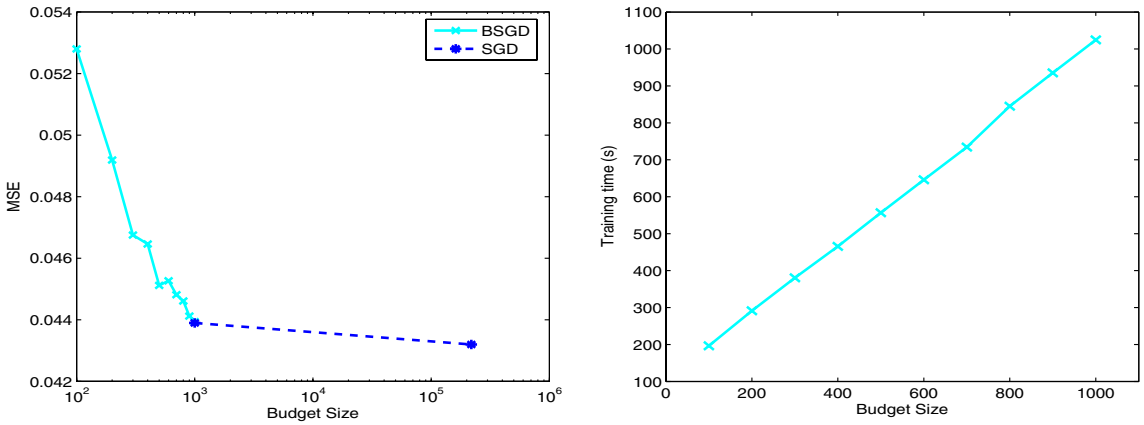

Fig. 10 MSE and training time of CASP with different budget size

Acknowledgements This work is supported by the National Natural Science Foundation of China under Grant Nos. 61432011, U1435212, and 61105054. This article is also supported by Commission for Collaborating Research Program, Key Laboratory of Solar Activity, National Astronomical Observatories, Chinese Academy of Sciences with KLSA201403.

References

1. Boser BE, Guyon IM, Vapnik VN (1992) A training algorithm for optimal margin classifiers. In: Proceedings of the fifth annual workshop on Computational learning theory, ACM, pp 144–152
2. Brown JD, Summers MF, Johnson BA (2015) Prediction of hydrogen and carbon chemical shifts from rna using database mining and support vector regression. J Biomol NMR 63(1):1–14
3. Chen J, Xue X, Ha M, Yu D, Ma L (2014) Support vector regression method for wind speed prediction incorporating probability prior knowledge. Math Probl Eng 2014(2014):1–10
4. Osuna E, Freund R, Girosi F (1997) Training support vector machines: an application to face detection. In: Computer Vision and Pattern Recognition, 1997. Proceedings, 1997 IEEE Computer Society Conference on 1997, pp 130–136
5. Ho CH, Lin CJ (2012) Large-scale linear support vector regression. J Mach Learn Res 13(1):3323–3348
6. Lin CJ, Weng RC, Keerthi SS (2007) Trust region newton method for large-scale logistic regression. J Mach Learn Res 9(2):561–568
7. Hsieh CJ, Chang KW, Lin CJ, Keerthi SS, Sundararajan S (2008) A dual coordinate descent method for large-scale linear SVM. In: ICML, pp 1369–1398
8. Xie X, Chen C, Chen Z (2015) Mini-batch quasi-newton optimization for large scale linear support vector regression. In: International Conference on Mechatronics, Materials, Chemistry and Computer Engineering
9. Wang Y, Ou G, Pang W, Huang L, Coghill GM (2016) e-distance weighted support vector regression. CoRR. arXiv:1607.06657
10. Wang Z, Crammer K, Vucetic S (2012) Breaking the curse of kernelization: budgeted stochastic gradient descent for large-scale SVM training. J Mach Learn Res 13(1):3103–3131
11. Zheng S (2015) A fast algorithm for training support vector regression via smoothed primal function minimization. Int J Mach Learn Cybern 6(1):1–12
12. Lu J, Hoi SC, Wang J, Zhao P, Liu Z-Y (2016) Large scale online kernel learning. J Mach Learn Res 17(47):1–43
13. Crammer K, Kandola J, Singer Y (2003) Online classification on a budget. Adv Neural Inf Process Syst 40(2):225–232
14. Cavallanti G, Cesa-Bianchi N, Gentile C (2006) Tracking the best hyperplane with a simple budget perceptron. Mach Learn 69(2–3):143–167
15. Dekel O, Shalev-Shwartz S, Singer Y (2008) The forgetron: a kernel-based perceptron on a budget. SIAM J Comput 37(5):1342–1372
16. Orabona F, Keshet J, Caputo B (2009) Bounded kernel-based online learning. J Mach Learn Res 10(6):2643–2666
17. Lvesque JC (2013) Ensembles of budgeted kernel support vector machines for parallel large scale learning. In: NIPS 2013 Workshop on Big Learning: Advances in Algorithms and Data Management
18. Smola AJ, Scholkopf B (2004) A tutorial on support vector regression. Stat Comput 14(3):199–222
19. Cristianini N, Scholkopf B (2002) Support vector machines and kernel methods: the new generation of learning machines. Ai Mag 23(3):31–41
20. Bordes A, Bottou L, Gallinari P (2009) Sgd-qn: careful quasi-newton stochastic gradient descent. J Mach Learn Res 10(3):1737–1754
21. Zhu ZA, Chen W, Wang G, Zhu C, Chen Z (2009) P-packSVM: Parallel Primal grAdient desCent Kernel SVM. In: IEEE 13th International Conference on Data Mining, pp 677–686
22. Kivinen J, Smola AJ, Williamson RC (2004) Online learning with kernels. IEEE Trans Signal Process 52:2165–2176
23. Shalev-Shwartz S, Singer Y, Srebro N, Cotter A (2007), Pegasos: primal estimated sub-gradient solver for SVM. In: Machine Learning, Proceedings of the Twenty-Fourth International Conference, pp 3–30
24. Chang CC, Lin CJ (2011) Libsvm: a library for support vector machines. ACM Trans Intell Syst Technol 2(3):389–396