Ridge-Adjusted Slack Variable Optimization for Supervised Classification

2013 IEEE INTERNATIONAL WORKSHOP ON MACHINE LEARNING FOR SIGNAL PROCESSING, SEPT. 22–25, 2013, SOUTHAMPTON, UK

RIDGE-ADJUSTED SLACK VARIABLE OPTIMIZATION FOR SUPERVISED CLASSIFICATION

Yinan Yu[a,c], *Konstantinos I. Diamantaras*[b], *Tomas McKelvey*[a], *S.Y. Kung*[c]

Chalmers University of Technology [a]
Gothenburg, Sweden
{yinan,tomas.mckelvey}@chalmers.se

TEI of Thessaloniki [b]
Thessaloniki, Greece
kdiamant@it.teithe.gr

Princeton University [c]
Princeton, USA
kung@princeton.edu

ABSTRACT

This paper presents an iterative classification algorithm called Ridge-adjusted Slack Variable Optimization (RiSVO). RiSVO is an iterative procedure with two steps: (1) A working subset of the training data is selected so as to reject "extreme" patterns. (2) the decision vector and threshold value are obtained by minimizing the energy function associated with the slack variables. From a computational perspective, we have established a sufficient condition for the "inclusion property" among successive working sets, which allows us to save computation time. Most importantly, under the inclusion property, the monotonic reduction of the energy function can be assured in both substeps at each iteration, thus assuring the convergence of the algorithm. Moreover, ridge regularization is incorporated to improve the robustness and better cope with over-fitting and ill-conditioned problems. To verify the proposed algorithm, we conducted simulations on three data sets from the UCI database: adult, shuttle and bank. Our simulation shows stability and convergence of the RiSVO method. The results also show improvement of performance over the SVM classifier.

Index Terms— slack energy minimization, kernel method, ridge-regression, classification, training data selection

1. INTRODUCTION

Support Vector Machine (SVM) [1, 2] is the optimal classifier in terms of the maximum margin between two classes. Classification techniques by Mean Squared Slack (MSS) minimization are closely related to SVM. The relation has been described in detail [3, 4] . In the previous work, a classification technique called Slackmin is presented. This approach attempts to minimize the MSS, which yields a much simpler computation compared to SVM for big data scenario. Instead of quadratic programming, the computational complexity for Slackmin is dominated by the inversion of the data covariance matrix, which mainly depends on the dimensionality of the feature vector. A kernel version Slackmin is also presented. The classification accuracy turns to be slightly better compared to SVM. In this paper, we are aiming to tackle the following issues:

(1) The slack variable associated with pattern i is defined as $\xi(i) = \max\{\gamma - t(i)(\boldsymbol{w}^T\boldsymbol{x}_i + b), 0\}$. Namely, all the "bad" patterns are taken into consideration, including also the outliers which could be detrimental to training.

(2) Slackmin evaluates all the patterns in the training set after each iteration to identify the active training set for the next step. In big data scenario, the computation can be costly.

(3) Over-training is inevitable for kernel Slackmin since there is no constraints on the regression coefficients.

(4) To make the algorithm more amenable to big data problem, parallelization of the algorithm will be explored.

The paper is organized as follows. First, it starts with a brief review of the previous work. Then in Section 3, basic concepts of the newly proposed techniques are established by introducing theoretical foundations. Details of the algorithms are then presented in Section 4 and experimental results are compared in Section 5.

2. PREVIOUS WORK

Recently, the "Slackmin" classification algorithm has been proposed [3, 4] based on the minimization of the slack variable energy. We briefly review this work and introduce some terminology in this section. More details can be found in the previous papers.

First, we define the slack variable associated with pattern $\boldsymbol{x}_i$ as follows:

$$\xi(i) = \max\{\gamma - t(i)(\boldsymbol{w}^T\boldsymbol{x}_i + b)^2, 0\} \tag{1}$$

where $\boldsymbol{w}$ is the weight vector and b the bias parameter. The objective of Slackmin is to minimize the Mean Squared Slack J_{MSS} defined as:

$$J_{MSS} \quad = \quad \frac{1}{2}E\{\xi^2 \mid \xi > 0\} \tag{2}$$

For any given pair $(\boldsymbol{w}, b)$ define the *active set*

$$S = \{\boldsymbol{x}_i : t(i)(\boldsymbol{w}^T\boldsymbol{x}_i + b) < \gamma, \forall i\} \tag{3}$$

as the set of patterns with positive slack variable $\xi(i)$, and denote the cardinality of S by $|S|$.

If we fix the active set S, the classical least squares solution for

$$\boldsymbol{\beta}^* = \begin{bmatrix}\boldsymbol{w}^* \\ b^*\end{bmatrix} = \arg\min_{\boldsymbol{w},b} J_{MSS} \tag{4}$$

problem is typically solved by zeroing the derivatives $\frac{\partial J_{MSS}}{\partial \boldsymbol{w}}$ and $\frac{\partial J_{MSS}}{\partial \boldsymbol{b}}$, which leads to

$$\boldsymbol{\beta}^* = \gamma \begin{bmatrix}\boldsymbol{R}_x & \boldsymbol{m}_x \\ \boldsymbol{m}_x^T & 1\end{bmatrix}^+ \begin{bmatrix}\boldsymbol{m}_{tx} \\ \boldsymbol{m}_t\end{bmatrix} \tag{5}$$

where $t(i) = \pm 1$, $\boldsymbol{R}_x = \frac{1}{|S|}\sum_{\boldsymbol{x}_i \in S} \boldsymbol{x}_i\boldsymbol{x}_i^T$, $\boldsymbol{m}_t = \frac{1}{|S|}\sum_{\boldsymbol{x}_i \in S} t(i)$, $\boldsymbol{m}_x = \frac{1}{|S|}\sum_{\boldsymbol{x}_i \in S} \boldsymbol{x}_i$, and $\boldsymbol{m}_{tx} = \frac{1}{|S|}\sum_{\boldsymbol{x}_i \in S} t(i)\boldsymbol{x}_i$.

Since S is not known, an iterative approach is employed for its computation. At each step k, we alternate between

(a) finding (given the active set S^k) the optimal weights $\boldsymbol{w}_k^*$ and bias b_k^* according to Eq. (5), and

(b) finding (given $\boldsymbol{w}_k^*$ and b_k^*) the next active set S^{k+1} by Eq. (3).

The initial condition $\left[\boldsymbol{w}_1^T,\, b_1\right]^T$ is randomly selected.

The algorithm can be naturally extended to the kernel case. As usual, we consider a nonlinear mapping $\Phi(\cdot)$ from the original feature space to a higher dimensional space and we assume that $\boldsymbol{w} = \sum_i a_i \Phi(\boldsymbol{x}_i)$. Instead of computing $\boldsymbol{\beta}$ by Eq. (5) we can find $\boldsymbol{\alpha} = [\boldsymbol{a}^T,\, b]^T$ as follows:

$$\boldsymbol{\alpha}^* = \begin{bmatrix} \boldsymbol{a}^* \\ b^* \end{bmatrix} = \gamma\, [\boldsymbol{K},\, \boldsymbol{e}]^+\, \boldsymbol{t}_S \qquad (6)$$

where $\boldsymbol{K}$ is the kernel matrix, whose entries are defined by a user chosen kernel function $k(x, y)$, such that $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ [6, 7, 8, 9]. Vector $\boldsymbol{e}$ is a column vector with all ones and $\boldsymbol{t}_S$ is the vector containing all targets $t(i)$, for $\boldsymbol{x}_i \in S$.

Let us call "the set which is not active" at step k the "complementary set" and denote it as $S_c^k = S_{all} \setminus S^k$, where S_{all} denotes the whole training data set. We also assign index k to J_{MSS}^k, $\boldsymbol{R}_x^k$, $\boldsymbol{m}_t^k$, $\boldsymbol{m}_x^k$ and $\boldsymbol{m}_{tx}^k$ to track the update when it is necessary.

3. THEORETICAL FOUNDATION FOR RISVO ALGORITHM

In this section, we first present the theoretical foundations (including selection of the active set and the main convergence theorem) for the proposed algorithm. Then, its ridge extension will be presented at the end of the section.

3.1. Active subset selection

Let

$$y(i) = \boldsymbol{w}^T \boldsymbol{x}_i + b \qquad (7)$$

for the linear kernel, and

$$y(i) = \sum_{j \in G} \boldsymbol{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)\boldsymbol{a}(j) + b \qquad (8)$$

for nonlinear kernels, where $G \subseteq S_{all}$. The new slack variables are defined as:

$$\xi(i) = \begin{cases} 1 - y(i)t(i) & \gamma_1 < y(i)t(i) < \gamma_2 \\ 0 & \text{otherwise} \end{cases} \qquad (9)$$

At step k, the patterns $\boldsymbol{x}_i$ associated with the nonzero $\xi(i)$ construct the active set S^k.

Compared to the previous method Slackmin [3], the new algorithm is different in the following way:

a. In Slackmin, the classifier is updated based on all the "bad" patterns. The term "bad" patterns referred to those cannot be correctly classified, i.e. those associated with $\xi(i) \geq 0$. However, the performance may be adversely affected by including the "extreme" patterns, i.e. those patterns with $\xi(i) \gg 0$, as they may very well possibly be outliers. One way of solving this is to discard as well the patterns associated with very large ξ. Namely, instead of one parameter γ, we could define boundaries (γ_1 and γ_2) on both sides for ξ.

b. In the previous algorithm, the expression of $\xi(i) = \gamma - t\left[\boldsymbol{w}^T \boldsymbol{x}_i + b\right]$ depends on γ. When we update $\boldsymbol{\beta}$ according to Equation (5) or (6), the effect of γ will be canceled in the next update. Therefore, we replace γ by 1 in the definition of $\xi(i)$.

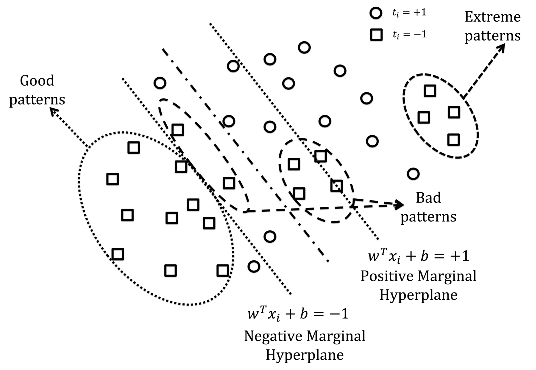


Fig. 1. An intuitive example of the active set selection. The difference between Slackmin and our new algorithm is that we exclude also the "extreme patterns" from training at each step.

A two-dimensional example is shown in Figure 1. We call $\boldsymbol{w}^T \boldsymbol{x} + b = 0$ the separation hyperplane and $\boldsymbol{w}^T \boldsymbol{x} + b = \pm 1$ the positive and negative marginal hyperplanes. As illustrated in the figure, the slack variables involved in the computations at each step are chosen to be the "bad patterns" in "Slackmin". However, according to our new definition , the "extreme patterns" are also ruled out.

3.2. Inclusion property and convergence

A sufficient condition for guaranteeing the convergence of the algorithm is that the sequence of active sets obeys the inclusion property defined below.

Definition 1. *Inclusion property of active set S:*

Given a sequence of active sets $\{S^k\}$, where $k = 1, \cdots, K$ and K is the maximum iteration number. We say the sequence has the **inclusion property** *if the following holds:*

$$S^1 \supsetneq S^2 \cdots \supsetneq S^k \cdots \supsetneq S^K \qquad (10)$$

i.e. once a pattern is removed at any time, then it will not be included in any of the future steps.

In addition to guaranteeing convergence, the inclusion property provides possibilities for faster computations and smaller memory requirements. The reason is that the inclusion property allows us to search in a subset of the previous active set at each step instead of going through the whole training set. A sufficient condition for such property is presented in Lemma 1. Note that we only discuss the linear case. Similar results can be extended to nonlinear kernels.

Lemma 1. *Sufficient condition for inclusion property:*

Let C_{-1}, C_{+1} denote the set of patterns from $t = -1$ and $t = +1$ class, respectively. Suppose training data from set C_t is bounded within a ball centered at $\boldsymbol{m}_x^t$ with radius R_t. Namely,

$$\|\boldsymbol{x}_i^t - \boldsymbol{m}_x^t\|_2 \le R_t \tag{11}$$

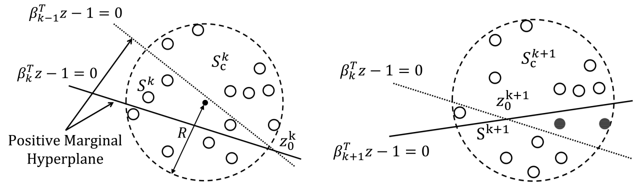
for all $\boldsymbol{x}_i^t \in C_t$ and $\boldsymbol{m}_x^t$ is the mean value of $\boldsymbol{x}_i^t$.

Let the active set $S^1 = S_{all}$ and $S^k = \{\boldsymbol{x}_i : t(i)(\boldsymbol{w}_k^T \boldsymbol{x}_i + b_k) < \gamma, \forall i\}$. Let $\boldsymbol{M}_k = \begin{bmatrix} \boldsymbol{\beta}_k^T \\ \boldsymbol{\beta}_{k+1}^T \end{bmatrix}$. The inclusion property of $\{S^k\}$ holds if we have:

$$\left\| \boldsymbol{M}_k (\boldsymbol{M}_k^T \boldsymbol{M}_k)^{-1} \begin{bmatrix} t \\ t \end{bmatrix} - \begin{bmatrix} t\boldsymbol{m}_x^{t,k} \\ t \end{bmatrix} \right\|_2 \ge R_t, \quad \forall k, t \tag{12}$$

Proof. Without loss of generality it suffices to study the positive class. Similar results can be derived for negative instances.

Let us construct a vector space $\mathcal{Z}$ by concatenating the feature vector $\boldsymbol{x}$ and its label t ($t = +1$ in this case): $\boldsymbol{z} = \begin{bmatrix} \boldsymbol{x} \\ 1 \end{bmatrix}$. The hyperplanes defined by the optimizer $\boldsymbol{\beta}_k^*$ at step k and $k+1$ intersect at points $\boldsymbol{z}_0^{k+1}$. A sufficient condition for inclusion property to hold is that $\boldsymbol{z}_0^{k+1}$ is inside the boundary of the training patterns.



An intuitive example can be found in the figure above. At step k, the intersection is located outside the ball which guarantees that $S^k \subsetneq S^{k-1}$. However, at step $k+1$, $\boldsymbol{z}_0^{k+1}$ is inside the boundary which allows the existence of the shallowed patterns. This violates $S^{k+1} \subsetneq S^k$ and hence the inclusion property can not be fulfilled. Note that our derivation does not depend on the two dimensional illustration. The example is just to give an intuitive understanding.

The intersection $\boldsymbol{z}_0^{k+1}$ can be expressed as points $\boldsymbol{z}$ such that:

$$\begin{bmatrix} \boldsymbol{\beta}_k^T \\ \boldsymbol{\beta}_{k+1}^T \end{bmatrix} \boldsymbol{z} = \boldsymbol{M}_k \boldsymbol{z} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{13}$$

Hence it can be derived as follows.

$$\hat{\boldsymbol{z}}_0^{k+1} = \boldsymbol{M}_k (\boldsymbol{M}_k^T \boldsymbol{M}_k)^{-1} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{14}$$

Note that full rank is assured for $\boldsymbol{M}$ when $k < K$.

A sufficient condition for the intersection being inside the boundary of class $t = +1$ is thus:

$$\left\| \hat{\boldsymbol{z}}_0^{k+1} - \begin{bmatrix} \boldsymbol{m}_x^{+1,k} \\ 1 \end{bmatrix} \right\|_2 \ge R_{+1}, \quad \forall k \tag{15}$$

□

If the inclusion property holds, it yields faster computations and requires less memory storage. Since S^{k+1} is always a subset of S^k, we do not need to evaluate the whole training patterns to select the active set at each step and the global convergence is straightforward. Therefore, one usage of Lemma 1 is that at each step k, if the sufficient condition is fulfilled, we can safely assume inclusion property and restrict our searching space at step $k+1$ to be a subset of S^k. Moreover, in practice, the inclusion property holds in most of the cases. Exceptions exist, yet $|S^{k+1} \setminus S^k| \to 0$ always holds for all k.Therefore, in our study, the inclusion property is always assumed. Consequently, in the new approach, at each iteration, we only select S^{k+1} as a subset of S^k.

The next step is to study the convergence of the algorithm. First, we define the empirical estimate of J_{MSS} called the *squared slack energy* as follows:

Definition 2. *Let $\xi(i)$ be a slack variable and S the active set. The squared slack energy is defined as:*

$$J_{SS} \;=\; \frac{1}{2} \sum_{i \in S} \xi(i)^2 \tag{16}$$

It can be easily seen that all previous mathematical results still hold if expectations are replaced by their corresponding empirical estimates. Therefore, we can conclude a sufficient condition for convergence as follows. Note that here the convergence is studied for the Slackmin algorithm, but it is valid for the new proposed approach as well. The reason is that in the new approach, we reject the outliers whose existence does not affect the outcome of the algorithm since they are not supposed to be involved in the training anyway.

Theorem 1. *Given a sequence of active sets $\{S^k\}$, the Slackmin Algorithm converges if the inclusion property holds.*

Proof. Let K be the maximum step number of the algorithm. We want to show that $J_{SS}^1 > J_{SS}^2 > \cdots > J_{SS}^K$.

We adopt the following notations:

a) $J_{SS}^{k*} = \frac{1}{2}\sum_{i \in S^k}(1 - \boldsymbol{\beta}_k^{*T}\boldsymbol{x}_i)^2$, i.e. the minimized squared slack energy at step k.

b) $J_{SS}^{k+1} = \frac{1}{2}\sum_{i \in S^{k+1}}(1 - \boldsymbol{\beta}_k^{*T}\boldsymbol{x}_i)^2$, i.e. the squared slack energy at step $k+1$ before minimization:

c) $J_{SS}^{(k+1)*} = \frac{1}{2}\sum_{i \in S^{k+1}}(1 - \boldsymbol{\beta}_{k+1}^{*T}\boldsymbol{x}_i)^2$, i.e. the minimized squared slack energy at step $k+1$.

It can be shown that

1) $J_{SS}^{k*} \le J_{SS}^{k+1}$: Under inclusion property $S^{k+1} \subsetneq S^k$, we know that 1) holds because we are only removing patterns from the training set, so the total energy is decreasing.

2) $J_{SS}^{k+1} \le J_{SS}^{(k+1)*}$ (because $\boldsymbol{\beta}_{k+1}^*$ minimizes J_{SS}^{k+1} at step $k+1$.

$k = 1 \cdots K$, $a) > b)$ and $b) > c)$). Combining 1) and 2), we conclude that $J_{SS}^{k*} \ge J_{SS}^{(k+1)*}$.

Since this holds for all k, the sequence $J_{SS}^1, J_{SS}^2, \cdots, J_{SS}^K$ is monotonically decreasing to a minimum value.

□

3.3. Ridge-adjusted extension

The *Ridge trace* [10, 11] in regression techniques penalizes the size of the regression coefficients and therefore helps with overfitting problems. In this section, ridge-adjusted extension of the algorithm is presented. This is one of the features differentiate Slackmin and our new algorithm.

3.3.1. *Linear case*

For the linear case, the extension to ridge regression is straightforward. We just add a ridge trace to the regressor as follows:

$$\boldsymbol{\beta}_k^* = \gamma \begin{bmatrix} \boldsymbol{R}_x + \rho \boldsymbol{I} & \boldsymbol{m}_x \\ \boldsymbol{m}_x^T & 1 \end{bmatrix}^+ \begin{bmatrix} \boldsymbol{m}_{tx} \\ \boldsymbol{m}_t \end{bmatrix} \tag{17}$$

3.3.2. *Kernel case*

Following [3] we call G the subset of S_{all}, which serves as the basis in the kernel computations:

$$y(i) = \sum_{j \in G} \boldsymbol{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)\boldsymbol{a}(j) + b \tag{18}$$

Without ambiguity, we use the notation $\boldsymbol{G}$ to represent the basis matrix containing $\boldsymbol{x}_{i \in G}$ as its columns. Besides, we use the following notations for further computations:

- Data vector in the kernel space: $\boldsymbol{\phi}(\boldsymbol{x})$.
- Data matrix in the kernel space: $\boldsymbol{\Phi}(\boldsymbol{x}) = [\boldsymbol{\phi}(\boldsymbol{x}_1), \cdots, \boldsymbol{\phi}(\boldsymbol{x}_{|S|})]$.
- Kernel matrix on the active set S: $\boldsymbol{K}_S = \boldsymbol{\Phi}(\boldsymbol{x})^T\boldsymbol{\Phi}(\boldsymbol{x})$, $\boldsymbol{x} \in S$.
- Kernel matrix on the basis set G: $\boldsymbol{K}_G = \boldsymbol{\Phi}(\boldsymbol{x})^T\boldsymbol{\Phi}(\boldsymbol{x})$, $\boldsymbol{x} \in G$.
- Inner product in the kernel space: $\boldsymbol{K}_{SG} = \boldsymbol{\Phi}(\boldsymbol{x})^T\boldsymbol{\Phi}(\boldsymbol{y})$, for $\boldsymbol{x} \in S$ and $\boldsymbol{y} \in G$.
- Vector with all ones: $\boldsymbol{e} = [1, \cdots, 1]^T$.
- Matrix with all ones: $\boldsymbol{E}$.

When we have the same size for the basis G and the active set S, ridge extension is obvious to show. However, due to the computational complexity, it commonly happens that we choose G to be a proper subset of S. In this case, the kernel matrix is no longer square, and hence ridge regression has to adapt accordingly with respect to the kernel tricks applied. We discuss the two cases as follows:

Case $G = S$:

$$\boldsymbol{\alpha}_k^* = \begin{bmatrix} \boldsymbol{a}^* \\ b^* \end{bmatrix} = \gamma\,[\boldsymbol{K}_S + \rho\boldsymbol{I} \quad , \quad \boldsymbol{e}]^+\,\boldsymbol{t}_S \tag{19}$$

Case $G \subsetneq S$:

$$\boldsymbol{\alpha}_k^* = \begin{bmatrix} \boldsymbol{a}^* \\ b^* \end{bmatrix} = \boldsymbol{C}_k^+(\boldsymbol{K}_{GS} + \boldsymbol{E})\boldsymbol{t}_S, \tag{20}$$

where $\boldsymbol{C}_k = [(\boldsymbol{K}_{GS} + \boldsymbol{E})\boldsymbol{K}_{SG} + \rho\boldsymbol{K}_G, \quad \boldsymbol{K}_{GS}\boldsymbol{e} + N_S\boldsymbol{e}]$.

Note that the purpose of selecting a subset $G \subsetneq S$ as the basis is to reduce the computational cost. Instead of producing a $|S| \times |S|$ kernel matrix, we only compute the matrix with size $|S| \times |G|$. Furthermore, by changing the size of the basis $|G|$, the classification performance varies. This has been studied in Section 5.

4. SEQUENTIAL AND PARALLEL RISVO

4.1. Sequential RiSVO algorithm

The algorithm we have introduced so far is called Ridge-adjusted Slack Variable Optimization (RiSVO). It is summarized in Algorithm RiSVO.

Algorithm RiSVO

```
- Initialization: $S_1 = S_{\text{all}}$
- For $k = 1 : K$
    - Switch kernel
        Case linear
            Update $\boldsymbol{\beta}_k$ according to Equation (17).
            Compute $y(i)t(i)$ from Equation (7)
            for all $i \in S^k$.
        Case nonlinear
            Update $\boldsymbol{\alpha}_k$ according to Equation (19)
            or (20).
            Compute $y(i)t(i)$ from Equation (8)
            for all $i \in S^k$.
    - end
    - Identify $S^{k+1} = \{\boldsymbol{x}_i : \gamma_1 < y(i)t(i) < \gamma_2\}$, where
      $\gamma_1$ and $\gamma_2$ are predefined
      (can be set as $\gamma_1 = -1$ and $\gamma_2 = 1$).
- end
```

4.2. Parallelization of RiSVO

In this work, parallelization of RiSVO is simulated using a sequential process implemented by a 'for' loop in Matlab. The simulated scheme is the well known technique called MapReduce. The MapReduce framework has been proposed in 2004 [12]. It allows simultaneous computation to reduce the complexity on each computing unit. Such a computing unit is called a node in this context. There are two steps in this framework: Map (dividing the work) and Reduce (merging the results). Many machine learning techniques adopt this concept to cope with big data problems [14, 13].

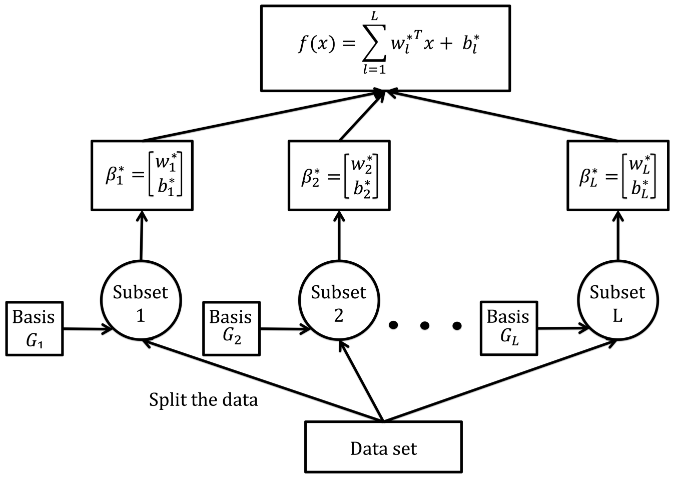

$f(x) = \sum_{l=1}^{L} w_l^{*T} x + b_l^*$
$\beta_1^* = \begin{bmatrix} w_1^* \\ b_1^* \end{bmatrix}$
$\beta_2^* = \begin{bmatrix} w_2^* \\ b_2^* \end{bmatrix}$
$\beta_L^* = \begin{bmatrix} w_L^* \\ b_L^* \end{bmatrix}$
Basis G_1
Subset 1
Basis G_2
Subset 2
Basis G_L
Subset L
Split the data
Data set

Fig. 2. MapReduce for parallel RiSVO. The data set is divided into L nodes. Each node carries out $\frac{1}{L}$ of the computation. The solutions computed from all the nodes are then combined in a linear fashion.

MapReduce is straightforward to apply for RiSVO. An illustrative example can be found in Figure 2. At the Map step, we simply divide the data set into L nodes and distribute the iterative computation to each node. At the Reduce step, the computed classification parameters are combined in a linear fashion to obtain the final classifier.

4.3. Choice of G

The choice of G may affect the generalization performance. From our experiences:

- When $|S_{\text{all}}|$ is reasonably large, let $|G| = |S_{\text{all}}|$. However, when the computation of a $|S_{\text{all}}| \times |S_{\text{all}}|$ kernel matrix is too heavy, choose $G \subsetneq S$.
- For parallel RiSVO, since the data are divided into L nodes, we have several possibilities of choosing G:
  a. Choose $G_1 = G_2 = \cdots = G_L$ and G_l a subset of S_{all}.
  b. Choose G_l as a random subset of S_l, $\forall l = 1, \cdots, L$.
  c. Choose G_1 as a random subset of S_{all} and G_{l+1} to be a subset of $S_{\text{all}}/(G_1 \cup \cdots \cup G_l)$, $\forall l = 1, \cdots, L-1$.

5. EXPERIMENTAL RESULTS

We have conducted simulations studies on three standard UCI data sets: adult, bank and shuttle. [5] The only preprocessing is to normalize each feature with its maximum value. In this section, we show the results obtained by sequential RiSVO and compare them with SVM [15] implemented by SVMlight [16]. Moreover, the results of parallel RiSVO is simulated by an independent sequential process. More results obtained by tuning some of the parameters can be found in Figure 3.

Data	Parameter	
Adult	$\rho = 0.0001$, $\lvert G\rvert = 30 \times dim$, RBF	
	Train accuracy	Test accuracy
	88.15 %	85.11 %
	Parameter	
Bank	$\rho = 0.0001$, $\lvert G\rvert = 30 \times dim$, RBF	
	Train accuracy	Test accuracy
	94.57 %	90.05 %
	Parameter	
Shuttle	$\rho = 0.0001$, $\lvert G\rvert = 30 \times dim$, RBF	
	Train accuracy	Test accuracy
	99.80 %	99.81 %

Table 2. The results of the simulated parallel RiSVO. The simulation is carried out by a sequence of independent computations in Matlab.

5.1. Sequential RiSVO

As we can see in Table 1, RiSVO outperforms SVM in most of the cases. For the data set shuttle, however, SVM with linear and rbf kernels work slightly better than the corresponding RiSVO algorithm. For fair comparison and consistency, we use the same kernel parameters for different methods, e.g. $\sigma = 1$ in rbf kernel and degree $d = 2$ of polynomial kernel. In our experiments, we select the basis matrix G according to method b. and the choice of $|G|$ depends on the dimension n of the feature vector. We choose $|G| = m \times n$, where $m \leq 50$ and m is an integer. The results shown in Figure 3 indicate that when $|G|$ is too small, $\boldsymbol{w}$ cannot be represented properly by too few samples. When $|G|$ is too large, the linear independence between the samples make it redundant to include all the data. Therefore, an optimal $|G|$ needs to be determined by cross validation.

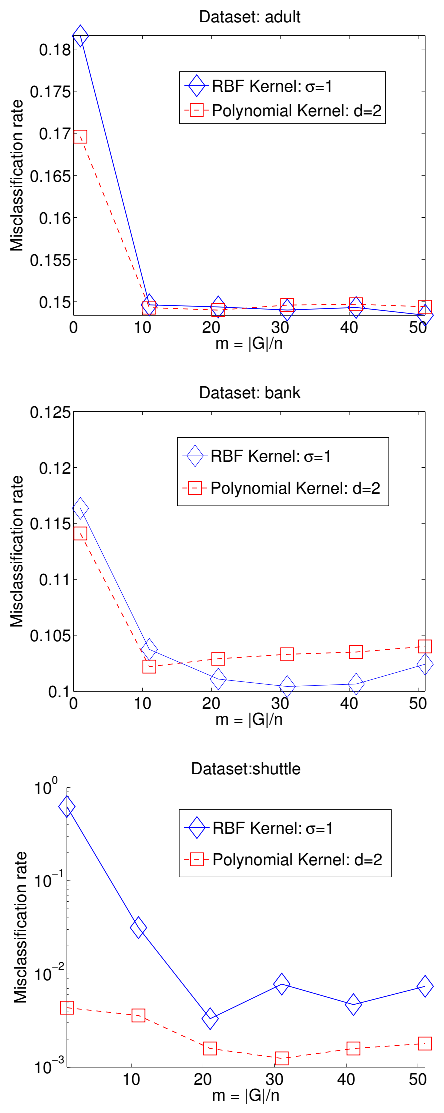


Fig. 3. Classification results of RiSVO with some different parameters versus the size of the basis $|\boldsymbol{G}|$.

5.2. Parallel RiSVO

The parallelization of RiSVO is simulated by a sequence of independent computations in Matlab using a "for" loop. The training and testing results are shown in Table 2.

Data	Dataset	Number of features		Number of training patterns		Number of testing patterns	
	Adult	14		32561		16281	
	Bank	16		45211		4521	
	Shuttle	9		43500		14500	
Classifier		Slackmin		SVM		RiSVO	
Dataset	Kernel	Parameter	Acc.%	Parameter	Acc.%	Parameter	Acc.%
Adult	LNR	$\times$	84.10	$C = 10$	84.12	$\rho = 0.0001$	84.43
	POLY	$d = 2$, $\lvert G\rvert = 560$	84.95	$C = 10$, $d = 2$	84.85	$\rho = 0.0001$, $d = 2$, $\lvert G\rvert = 560$	85.17
	RBF	$\sigma = 1$, $\lvert G\rvert = 560$	85.06	$C = 10$, $\sigma = 1$	84.97	$\rho = 0.0001$, $\sigma = 1$, $\lvert G\rvert = 560$	85.23
Bank	LNR	$\times$	88.83	$C = 10$	88.48	$\rho = 0.0001$	89.01
	POLY	$d = 2$, $\lvert G\rvert = 576$	89.69	$C = 10$, $d = 2$	88.98	$\rho = 0.0001$, $d = 2$, $\lvert G\rvert = 576$	89.69
	RBF	$\sigma = 1$, $\lvert G\rvert = 576$	90.47	$C = 10$, $\sigma = 1$	90.40	$\rho = 0.0001$, $\sigma = 1$, $\lvert G\rvert = 576$	90.60
Shuttle	LNR	$\times$	96.75	$C = 10$	97.56	$\rho = 0.0001$	97.39
	POLY	$d = 2$, $\lvert G\rvert = 450$	99.30	$C = 10$, $d = 2$	99.03	$\rho = 0.0001$, $d = 2$, $\lvert G\rvert = 450$	99.85
	RBF	$\sigma = 1$, $\lvert G\rvert = 450$	99.84	$C = 10$, $\sigma = 1$	99.78	$\rho = 0.0001$, $\sigma = 1$, $\lvert G\rvert = 450$	99.72

Table 1. In this table, simulations are conducted on three UCI datasets. The descriptions are given in the table. The proposed new algorithm RiSVO is compared with the related work (Slackmin algorithm) and SVM. In comparison, RiSVO provides a fast and accurate solution to classification of big datasets.

6. CONCLUSION

In this paper, we have proposed a new classification algorithm called Ridge-adjusted Slack Variable Optimization (RiSVO). Compared to the previous work, the active set is selected according to a different criterion for better flexibility. Furthermore, ridge regression is incorporated to maintain the robustness of the technique. By assuming inclusion property, RiSVO achieves a faster computations and convergence. Moreover, parallelization of RiSVO are simulated using sequential computations in Matlab, which allows RiSVO to tackle big data problems. In most of our experiments, RiSVO outperforms Slackmin and SVM with faster computations. Future work including implementation of parallel RiSVO, comparison of computational time, and RiSVO for multi-classifications are under progress. More parameter tuning and analysis will also be conducted at the next stage of the study.

7. ACKNOWLEDGMENT

This material is based on research sponsored by DARPA under agreement number FA8750-12-2-0126. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

This work is also sponsored by the Swedish Research Council (VR) which is gratefully acknowledged.

8. REFERENCES

[1] Vapnik, Vladimir N., *Statistical Learning Theory*, 1st Edition, Wiley-Interscience, September, 1998.

[2] Mavroforakis M., Theodoridis S., *A Geometric Approach to Support Vector Machine (SVM) Classification.* IEEE Transaction on Neural Networks, vol. 17(3), pp.671-683, 2006.

[3] Diamantaras K. I. and Kotti M., *Binary Classification By Minimizing the Mean Squared Slack*, Proceeding of IEEE ICASSP, 2012.

[4] Kotti M. and Diamantaras K. I., *Towards Minimizing the Energy of Slack Variables for Binary Classification*, Proceeding of 20th EUSIPCO, 2012.

[5] Bache, K. and Lichman, M. *UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]*, Irvine, CA: University of California, School of Information and Computer Science. 2013.

[6] Schlkopf, B. and Smola, A. J., *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond* 1st Edition, The MIT Press, Dec. 2001.

[7] Slavakis K., Theodoridis S., Yamada I., *Online classification using kernels and projection-based adaptive algorithms.* IEEE Transactions on Signal Processing, vol. 56(7), pp. 2781-2797, 2008.

[8] Bouboulis P. and Theodoridis S., *Extension of Wirtinger Calculus to Reproducing Kernel Hilbert Spaces and the complex kernel LMS.* IEEE Transactions on Signal Processing, vol. 53(3), pp. 964-978, 2011.

[9] Slavakis K., Bouboulis P., Theodoridis S., *Online Learning in Reproducing Kernel Spaces.* E-reference for Signal Processing, Elsevier, 2013.

[10] Hoerl A. E. and Kennard R. W. *Ridge regression: Biased estimation for nonorthogonal problems*, Technometrics, 42(1):80-86, 1970.

[11] Jain R.K., *Ridge regression and its application to medical data*, Computers and Biomedical Research Volume 18, Issue 4, pp. 363368, Aug. 1985.

[12] Jeffrey Dean and Sanjay Ghemawat, *MapReduce: Simplified Data Processing on Large Clusters*, 6th Symposium on Operating System Design and Implementation, San Francisco, CA, Dec., 2004.

[13] F. Ozgur Catak, and M. Erdal Balaban, *CloudSVM: Training an SVM Classifier in Cloud Computing Systems*, Pervasive Computing and the Networked World, Lecture Notes in Computer Science, Vol. 7719, pp 57-68, 2013.

[14] Chu C., Kim S. K., Lin Y., Yu Y., Bradski G., Ng A. Y. and Olukotun K., *Map-Reduce for Machine Learning on Multicore*, proceeding of NIPS, Dec. 2006.

[15] Cristianini, N. and Shawe-Taylor, J., *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*, 1st Edition, Cambridge University Press, March, 2000.

[16] Joachims T., *Making large-Scale SVM Learning Practical*, Advances in Kernel Methods - Support Vector Learning, B. Schölkopf and C. Burges and A. Smola (ed.), MIT-Press, 1999.