Parameter Tuning via Kernel Matrix Approximation for Support Vector Machine

Chenhao Yang, Lizhong Ding and Shizhong Liao
School of Computer Science and Technology / Tianjin University, Tianjin, China
Email: szliao@tju.edu.cn

Abstract—Parameter tuning is essential to generalization of support vector machine (SVM). Previous methods usually adopt a nested two-layer framework, where the inner layer solves a convex optimization problem, and the outer layer selects the hyper-parameters by minimizing either cross validation or other error bounds. In this paper, we propose a novel parameter tuning approach for SVM via kernel matrix approximation, based on the observation that approximate computation is sufficient for parameter tuning. We first develop a preliminary approximate computation theory of parameter tuning for SVM. We present a kernel matrix approximation algorithm MoCIC. We design an approximate parameter tuning algorithm APT, which applies MoCIC to compute a low-dimension and low-rank approximation of the kernel matrix, and uses this approximate matrix to efficiently solve the quadratic programming of SVM, then selects the optimal candidate parameter through the approximate cross validation error (ACVE). Finally, we verify and compare the feasibility and efficiency of APT on 10 artificial and benchmark datasets. Experimental results show that this new algorithm can dramatically reduce time consumption of parameter tuning and at the same time guarantee the effectiveness of the selected parameters. It comes to the conclusion that the approximate parameter tuning approach is sound, efficient, and promising.

Index Terms—kernel methods, parameter tuning, support vector machine, matrix approximation

I. INTRODUCTION

Support vector machine (SVM) is a theoretically well motivated algorithm developed from statistical learning theory. It is a significant learning system for efficiently training the linear machines in the kernel-induced feature spaces, while controlling the capacity to prevent overfitting by generalization theory [1].

Parameter tuning has an essential influence on the generalization performance of SVM. The traditional methods mainly adopt a nested two-layer framework [2] for choosing the optimal parameter, where the inner layer solves a convex optimization problem to obtain Lagrange multipliers for fixed values of the hyper-parameters including the kernel parameters and the penalty factor, and the outer layer adjusts the hyper-parameters by minimizing the estimates of generalization error, either cross validation [3]-[5] or other error bounds [6]-[8].

For 1-norm soft margin SVM, cross validation gives an excellent estimate of the generalization error [3], but it demand a grid search over the parameter space, which unavoidably brings high computational complexity [9] because the inner convex optimization problem has to be iterated more times. So far, several approaches have been proposed to improve the efficiency of grid search, such as genetic algorithms [4] and evolution computation [5]. Approximate error bounds can also be taken as the criteria to select hyper-parameters. The commonly used bounds include span bound, Jaakkola-Haussler bound [7] and radius-margin bound [7], [8]. Generally, these two approaches take several strategies to reduce the search space of hyper-parameters to accelerate the outer layer of parameter tuning, then the inner-layer computation could be reduced of great quantity. Even so the determination of the search direction is usually of high cost and it is hard to verify the effectiveness of the search direction, and one iteration of the inner-layer’s computation stays unchanged. Furthermore, the complexity of quadratic programming for solving SVM is $O(n^3)$ and that of second-order cone programming (SOCP) for multiple kernel SVM is $O(Nn^{3.5})$ [10], where $n$ is the size of training examples and $N$ is the number of candidate kernels. When it come to large-scale problems, it is prohibitive to directly train SVM for every candidate model, and it is desirable to design an algorithm that can improve the efficiency of parameter tuning based on the acceleration of the inner optimization.

Actually, for parameter tuning it is unnecessary to calculate the accurate criteria, such as the optimal value of the objective function, cross validation error and test set accuracy, and for every candidate parameter; it is sufficient to calculate the approximate criteria that can discriminate the optimal one among candidate parameters. Such considerations drive the proposal of approximate parameter tuning approach for SVM, from which we exploit the matrix approximation’s computational model. We first develop a kernel matrix approximation algorithm MoCIC which synthesizes Monte Carlo approximation algorithm [11], [12] and the incomplete Cholesky factorization algorithm[13]. Then we propose an approximate parameter tuning algorithm APT, which applies MoCIC to compute a low-rank approximation of the kernel matrix, uses the approximate matrix to efficiently solve the convex quadratic programming of SVM, and selects the optimal parameters through the

© 2012 ACADEMY PUBLISHER
doi:10.4304/jcp.7.8.2047-2054
approximate cross validation error (ACVE). Experiments on artificial and benchmark datasets suggest that APT can improve the efficiency of parameter tuning and at the same time guarantee the generalization performance of the selected parameters.

The rest of the paper is organized as follows. In Section 2, we give a brief introduction of SVM. In Section 3, we present the concept of kernel matrix approximation and also present the MoCIC. In Section 4, we propose the approximate parameter tuning algorithm APT and as well discuss its time complexity. In Section 5, experimental results show a comparison between parameter tuning algorithm with the original kernel matrix and with approximation of the kernel matrix gained by MoCIC. The last section gives the conclusion.

II. SUPPORT VECTOR MACHINE

We use \( \mathcal{X} \) to denote the input space and \( \mathcal{Y} \) the output domain. Usually we will have \( \mathcal{X} \subseteq \mathbb{R}^p, \mathcal{Y} = \{-1, 1\} \) for binary classification. The training set is denoted by

\[
S = \{(x_1, y_1), \ldots, (x_n, y_n)\} \in (\mathcal{X} \times \mathcal{Y})^n,
\]

where we refer to the \( x_i \) as examples and the \( y_i \) as their labels, and \( n \) is the number of instances.

The maximal margin classifier is the basic model of SVM. It chooses the hyperplane \((w, b)\) that maximizes the classification margin for the linearly separable data. By solving the optimization problem

\[
\min \langle w, w \rangle, \\
\text{s.t. } y_i (\langle w, x_i \rangle + b) \geq 1, \quad i = 1, \ldots, n,
\]

we can obtain the hyperplane \((w, b)\).

The equivalent dual form of formula (1) is

\[
\min \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle - \sum_{i=1}^{n} \alpha_i, \\
\text{s.t. } \sum_{i=1}^{n} y_i \alpha_i = 0, \\
\alpha_i \geq 0, i = 1, \ldots, n,
\]

where \( \alpha_i, \alpha_j \) are the Lagrange multipliers.

Implicitly mapping the training data into the feature space defined by kernel \( K(x, z) \), we can obtain the kernel version of formula (2):

\[
\min \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j K(x_i, x_j) - \sum_{i=1}^{n} \alpha_i, \\
\text{s.t. } \sum_{i=1}^{n} y_i \alpha_i = 0, \\
\alpha_i \geq 0, i = 1, \ldots, n.
\]

For the non-separable data in the feature space, we can obtain the optimum hyperplane by solving the following optimization problem,

\[
\min \frac{1}{2} \langle w, w \rangle + C \sum_{i=1}^{n} \xi_i, \\
\text{s.t. } y_i (\langle w, x_i \rangle + b) \geq 1 - \xi_i, \\
\xi_i \geq 0, i = 1, \ldots, n.
\]

When using kernel trick the equivalent dual form of formula (4) is

\[
\min \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j K(x_i, x_j) - \sum_{i=1}^{n} \alpha_i, \\
\text{s.t. } \sum_{i=1}^{n} y_i \alpha_i = 0, \\
C \geq \alpha_i \geq 0, i = 1, \ldots, n,
\]

where \( C \) is the penalty factor.

The formula (5) is equivalent to the following problem

\[
\min \frac{1}{2} \alpha^T Q \alpha - e^T \alpha, \\
\text{s.t. } y^T \alpha = 0, \\
C \geq \alpha \geq 0,
\]

where \( y \in \mathcal{Y}^n \) is the label vector, \( e \) is the \( n \)-vector of ones, the matrix \( Q \) is the matrix \( \{Q_{ij} = y_i y_j K(x_i, x_j)\} \), and inequality \( C \geq \alpha \geq 0 \) means \( C \geq \alpha_i \geq 0, i = 1, \ldots, n \).

Denoting the solutions of problem by \( \alpha^*_i, i = 1, \ldots, n \), the final hypothesis can be defined as

\[
f(x) = \text{sgn} \left( \sum_{i=1}^{n} \alpha^*_i y_i K(x_i, x) + b \right).
\]

From the above optimization formulae, we can find that the performance of SVM mainly depends on the Lagrange multipliers and the hyper-parameters. The Lagrange multipliers can be easily obtained through solving the quadratic programming when hyper-parameters are fixed, so the hyper-parameters have a decisive influence on the performance of SVM.

In this paper, we aim at developing an efficient method for the selection of hyper-parameters from the perspective of kernel matrix approximation.

III. KERNEL MATRIX APPROXIMATION

It is a fundamental result of linear algebra [14] that for any matrix \( A \) and positive integer \( k \), there exists a matrix \( A_k \) which simultaneously minimizes \( \| A - D \| \) over rank \( k \) matrices \( D \), for all norms that are invariant under rotation (for example, the Frobenius norm and the 2-norm). \( A_k \) is called the optimal rank \( k \) approximation, and its computation, follow from the singular value decomposition (SVD) of \( A \). However, the computational complexity of SVD is \( O(n^3) \) for \( A \in \mathbb{R}^{n \times n} \).
The objective of kernel matrix approximation is to design an efficient algorithm to obtain a near-optimal low-rank approximation of the kernel matrix satisfying
\[ \| A - \tilde{A}_k \| \leq \| A - \bar{A}_k \| + \mu, \]
where \( \mu \) represents a tolerable level of error for the given application.

A. Algorithm MoCIC

In this paper, synthesizing the Monte Carlo algorithm [11],[12] and incomplete Cholesky factorization [15], we develop a kernel matrix approximation algorithm MoCIC (Monte Carlo algorithm and incomplete Cholesky factorization), shown in Algorithm 1, which uses the Monte Carlo algorithm to randomly sample the kernel matrix and then applies the incomplete Cholesky factorization with symmetric permutation to obtain the near-optimal low-rank approximation of the low-dimension sample matrix.

In MoCIC, \( \{ p_i \}_{i=1}^n \) denotes the sampling probability distribution where \( p_i \) is the probability that the column \( i \) is chosen, satisfying \( \sum_{i=1}^n p_i = 1 \). Usually the sampling probabilities of the form \( p_i = 1/n \) or \( p_i = Q_n / \sum_{i=1}^n Q_n \)

or \( p_i = |Q(i)|^2 / \| Q \|^2 \) are used. \( c \) is the sampling size. \( \epsilon_{tol} \) is the lower bound of the trace of the matrix \( G \), through which we can directly tune the rank of the output matrix. \( S_i \) is the zero-one sampling matrix where \( S_{i,r} = 1 \) if the \( i \)-th column of \( Q \) is chosen in the \( r \)-th sampling, and \( D \) is the rescaling matrix.

B. Time Complexity

MoCIC can be divided into two parts. When given an \( n \times n \) matrix \( Q_{nn} \), MoCIC first obtains a \( c \times c \) matrix \( Q_{cc} \) by random sampling, the complexity is \( O(cn) \) [16]. As is well known, any positive definite matrix \( Q \) can be decomposed by Cholesky factorization with the form \( Q = GG^T \), where \( G \) is a lower triangular matrix. Then, MoCIC applies incomplete Cholesky factorization to get a near-optimal rank \( k \) approximation matrix \( GG^T \) of the sampling matrix \( Q_{cc} \), where rank\((GG^T) = \text{rank}(G) = k \).

The complexity is \( O(ck^3) \) [13]. Therefore, from above the total complexity of MoCIC is \( O(cn + ck^2 + c^2k) \), where \( O(c^2k) \) comes from matrix multiplication.

Reference [17] gives a method to approximate Gram matrix \( G \) which chooses \( c \) columns from \( G \) uniformly at random and without replacement, and constructs an approximation of the form \( \tilde{G} = CW^{-1}C^T \), where the \( n \times c \) matrix \( C \) consists of the \( c \) chosen columns and \( W \) is a matrix consisting of the intersection of those \( c \) columns with the corresponding \( c \) rows. This method has been referred to as the Nyström method [17]-[19] since it has an interpretation in terms of the Nyström technique for solving linear integral equations. And here we call this method as Preliminary Nyström algorithm. Based on this [12], a generalization of the Preliminary Nyström algorithm was raised which allows the column sample to be formed using arbitrary sampling probabilities. We will show this algorithm in Algorithm 2. Algorithm 2 takes as input an \( n \times n \) Gram matrix \( G \), a probability distribution \( \{ p_i \}_{i=1}^n \), a number \( c \leq n \) of columns to choose, and a rank parameter \( k \leq c \). It returns as output an approximate decomposition of the form \( \tilde{G}_k = CW_k^T C^T \), where \( C \) is an \( n \times c \) matrix consisting of the chosen columns of \( G \), each rescaled in an appropriate manner, \( W_k^T \) is the

---

Algorithm 1: MoCIC

**Input:** \( n \times n \) matrix \( Q \), \( \{ p_i \}_{i=1}^n \), \( c \), \( \epsilon_{tol} \).

**Output:** \( \tilde{Q} \).

If \( n > c \) then

1. \( S = \text{zeros}(n,c) \), \( D = \text{zeros}(c,c) \);
2. for \( r = 1 : c \) do
   1. Pick \( i_r \in \{1, \ldots, n\} \) with \( Pr[i_r = i] = p_i \);
   2. \( S_{i_r} = 1; D_{i_r} = 1 / \sqrt{p_i} \);
3. \( n = c; Q = DS^TQSD; \)
4. for \( i = 1 : n \) do
   1. for \( j = i : n \) do
      1. \( G_{jj} = Q_{jj} \);
      2. for \( k = 1 : i-1 \) do
         1. \( G_{jk} = G_{jj} - G_{jk}G_{jk} \);
      end
   end
   1. if \( \sum_{j=i}^n G_{jj} \leq \epsilon_{tol} \) then
      1. break;
   else
      1. \( G_{i+r} = \max_{j \in R} G_{jj} \);
      2. \( G_{r,t} = Q_{j_m,j} \);
      3. \( G_{r,t} \leftarrow G_{i,t} \);
      4. \( G_{i,t} = G_{i,t} + G_{i+r,t} / G_{ij} \);
      5. for \( j = 1 : i-1 \) do
         1. \( G_{i+r,j} = G_{i+r,j} - G_{i+r,t}G_{i,t} \);
      end
   end
end

Return \( \tilde{Q} = GG^T \).
Algorithm 2: Nyström Algorithm

**Input:** \( n \times n \) Gram matrix \( G \), \( \{p_i\}_{i=1}^n \) such that \( \sum_{i=1}^n p_i = 1 \).

**Output:** \( n \times n \) matrix \( \tilde{G} \).

- Define \( S = 0_{n \times c} \).
- Define \( D = 0_{c \times c} \).

**for** \( t = 1, \ldots, c \) **do**

- Pick \( i \in [n] \), where \( \Pr(i_t) = p_t \);
- \( D_t = (c_p_t)^{1/2}, S_{i=t} \);

**end**

Let \( C = GSD \);

Let \( W = DS^T GSD \);

Compute \( W_k \), the best rank-\( k \) approximation to \( W \);

Return \( \tilde{G}_k = CW_k^C W_k^T \).

Moore-Penrose generalized of \( W_k \), where \( W_k \) is a \( c \times c \) matrix that is the best rank-\( k \) approximation to matrix \( W \), which is a matrix whose elements consist of those elements in \( G \) in the intersection of the chosen columns and the corresponding rows, each rescaled in an appropriate manner. The SVD and the matrix inversion are essential steps in computing \( W_k \) to obtain the approximate matrix. Therefore, the total complexity of the algorithm is \( O(cn + c^2 + cn^2) \). It’s apparent that \( O(cn + c^2 + c^2k) \leq O(cn + c^2 + cn^2) \). Therefore, MoCIC will be more efficient.

**IV. APPROXIMATE PARAMETER TUNING**

**A. Algorithm APT**

Based on the MoCIC algorithm, we propose an approximate parameter tuning algorithm APT shown in Algorithm 3.

We use the Gaussian kernel

\[
k(x_i, x_j) = \exp(-\gamma \| x_i - x_j \|^2)
\]

to describe our algorithm, where \( \gamma = 1/2\sigma^2 \), but actually the framework of Algorithm 3 is suitable to any other kernels. First, we divide the data set \( S = \{(x_i, y_i)\}_{i=1}^n \) into training set and validation set. For every kernel parameter \( \gamma \), we use the training set and the corresponding labels to generate the kernel matrix \( Q \). Second, we apply MoCIC to compute the low-rank approximation \( \tilde{Q} \) of the kernel matrix \( Q \). Third, using \( \tilde{Q} \) to solve SVM through the quadratic programming, this procedure is efficient and approximate result is received. Finally, for every hyper-parameters setting \((\gamma, C)\), we can get an approximate cross validation error (ACVE). We will choose the hyper-parameters setting with the minimum ACVE as the final output \((\gamma, C)^*\).

In APT, we can set the Distype to be 1, 2 or 3, meaning that the sampling distribution is \( p_i = 1/n \), \( p_i = Q_{ii} / \sum_{i=1}^n Q_{ii} \) or \( p_i = |Q_{ii}|^2 / \| Q_{ii} \|^2 \). The folds of cross validation are usually set as \( t = 5 \). The input \( c, \epsilon_{tol} \) are the same as MoCIC. The Cinterval, Glinterval denote the tuning intervals of the hyper-parameters.

**B. Time Complexity**

Let \( S_\gamma \) and \( S_C \) denote the iteration steps of \( \gamma \) and \( C \).
The complexity of quadratic programming for solving SVM is \( O(n^3) \), \( n \) is the size of training set. Therefore,

Algorithm 3: APT

**Input:** \( S = \{(x_i, y_i)\}_{i=1}^n \), \( \gamma, C, \epsilon_{tol}, \) DisType, CInterval \( \in \mathbb{R}^3 \), Glinterval \( \in \mathbb{R}^3 \)

**Output:** \((\gamma, C)^*\)

(\(CBegin, CEnd, CStep\))=CInterval;

(\(GBegin, GEnd, GStep\))=Glinterval;

\(Q = zeros(n,n)\);

**for** \( fold = 1 : t \) **do**

\((TrainingSet, ValidationSet) = CVPartition(S, fold)\);

\(m = size(TrainingSet,1)\);

**for** \( i = GBegin : GStep : GEnd \) **do**

\(j = 1 : m \) **do**

**for** \( k = 1 : m \) **do**

Take \( (x_j, y_j), (x_k, y_k) \in TrainingSet \);

\(Q_{jk} = y_j y_k RBFKernel(2^j, x_j, x_k)\);

**end**

end

\(\{p_i\}_{i=1}^n = \)SetDistribution\((Q, DisType)\);

\(\tilde{Q} = MoCIC(Q, \{p_i\}_{i=1}^n, c, \epsilon_{tol})\);

**for** \( j = CBegin : CStep : CEnd \) **do**

\(DecisionFunction = SVC(\tilde{Q}, 2^j)\);

\(CError(2^j, 2^j, fold) =\)

\(SVCError(DecisionFunction, ValidationSet)\);

**end**

end

\((\gamma, C)^* = \arg \min_{(\gamma, C)} (CError(2^j, 2^j, 1 : t))\);

Return \((\gamma, C)^*\);
the complexity of \( t \)-fold cross validation for SVM model selection is \( O(tS_S c^3) \). APT applies MoCIC to obtain a \( c \times c \) approximate matrix \( \hat{Q} \). The complexity of kernel matrix approximation is \( O(cn + ck^2 + c^2k) \). The complexity of quadratic programming using \( \hat{Q} \) is \( O(c^3) \). The complexity of kernel matrix approximation \( O(cn + ck^2 + c^2k) \leq O(c^3) \) for \( c > \sqrt{n} \). Therefore, the total complexity of APT is \( O(tS_S c^3) \). For radius margin bound or span bound, let \( S_{gd} \) denote the iteration steps of gradient descent. For every iteration, a standard SVM need to be solved in the inner layer, so the total complexity of these methods is \( O(S_{gd} n^3) \). But the complexity \( O(S_{gd} n^3) \) prevents these methods from scaling to the large scale problem when \( n \) is very large. However the APT algorithm is scalable. In the following sections, a series of experiments conducted on different kinds of datasets can demonstrate the feasibility of this scaling.

V. EXPERIMENT

In this section we verify the feasibility and efficiency of APT.

The benchmark datasets used in our experiments are chosen from UCI, Statlog and Delve databases shown in Table I and the artificial dataset “Fourclass” is from [20]. All experiments are performed on a Core2 Quad PC, with 2.33GHz CPU and 4GB memory.

A. The Feasibility

We first examine the feasibility of APT.

We use the parameterization \( \Theta = (\log_2 C, \log_2 \gamma) \). For different datasets we set the different parameters tuning intervals as shown in Table I. We apply APT to different datasets under different sampling sizes, to observe the running time of APT and the test set accuracy (TSA) of the model produced by APT. The sampling sizes are set to be 0.2n, 0.4n, 0.6n, 0.8n, where \( n \) is the number of examples. With the sampling size decreases in a certain interval, if the TSA of models produced by APT decreases negligibly, the APT algorithm is feasible, or otherwise, it is unfeasible.

The results for different datasets are shown in Fig.1. We can find that with the sampling size decreases in a certain interval (namely \( [0.4n, 0.8n] \)), the running time of APT drops sharply but the changes of TSA is nearly negligible. The experimental results fully demonstrate the feasibility of APT.

B. The Efficiency

We further verify the efficiency of APT.

We set \( \epsilon_{tol} \) to be fixed value \( 10^{-12} \) and \( Distype = 3 \) in APT. For every hyper-parameters setting, we generate the kernel matrix \( Q \) using the Gaussian kernel, and then use MoCIC to compute the low-rank approximation matrix \( \hat{Q} \) of the kernel matrix \( Q \), and next input the matrix \( \hat{Q} \) to approximately and efficiently solve SVM through the quadratic programming, and finally take the hyper-parameters setting with the minimum approximate cross
validation error (ACVE) as the output. For different sampling size, we obtain the different approximation matrices and therefore we may get different approximate optimal models. The “approximate optimal” is evaluated by the ACVE. Table II shows optimal parameter $\gamma, C$, running time and the ACVE under different sampling size.

We take the parameter produced by APT under the sampling size $c = 0.4n$ as the approximate optimal parameter and the parameter produced by the original kernel matrix as the accurate optimal model. Table III shows the comparison between accurate and approximate optimal parameters of different datasets, where the subscript “acc” and “app” are the abbreviation for accurate and approximate. We compare the solving time and the 5-fold cross validation accuracy between these two different optimal parameters. The results are shown in Table IV. We can find that the time for obtaining the approximate optimal parameter is obviously less than that of the accurate optimal parameter and the time gap grows rapidly as the growth of dataset’s size. Nevertheless, the TSA between different optimal models are very close.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Sampling size</th>
<th>$\gamma$</th>
<th>$C$</th>
<th>Time (s)</th>
<th>ACVE (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sonar</td>
<td>41</td>
<td>0.125000</td>
<td>2</td>
<td>64.8</td>
<td>29.8</td>
</tr>
<tr>
<td></td>
<td>82</td>
<td>0.031250</td>
<td>16</td>
<td>132.9</td>
<td>16.2</td>
</tr>
<tr>
<td></td>
<td>123</td>
<td>0.062500</td>
<td>8</td>
<td>261.8</td>
<td>14.4</td>
</tr>
<tr>
<td>Heart</td>
<td>54</td>
<td>0.001953</td>
<td>4</td>
<td>181.2</td>
<td>19.3</td>
</tr>
<tr>
<td></td>
<td>108</td>
<td>0.000061</td>
<td>512</td>
<td>429.0</td>
<td>16.3</td>
</tr>
<tr>
<td></td>
<td>162</td>
<td>0.015625</td>
<td>2</td>
<td>961.1</td>
<td>14.4</td>
</tr>
<tr>
<td>Ionosphere</td>
<td>70</td>
<td>0.500000</td>
<td>1</td>
<td>169.0</td>
<td>23.4</td>
</tr>
<tr>
<td></td>
<td>140</td>
<td>0.500000</td>
<td>2</td>
<td>431.6</td>
<td>10.2</td>
</tr>
<tr>
<td></td>
<td>210</td>
<td>0.250000</td>
<td>1</td>
<td>1041.1</td>
<td>5.7</td>
</tr>
<tr>
<td>Breast</td>
<td>136</td>
<td>0.000977</td>
<td>1</td>
<td>660.0</td>
<td>3.2</td>
</tr>
<tr>
<td></td>
<td>272</td>
<td>0.007813</td>
<td>1</td>
<td>2334.8</td>
<td>2.5</td>
</tr>
<tr>
<td></td>
<td>408</td>
<td>0.003906</td>
<td>2</td>
<td>6394.2</td>
<td>2.3</td>
</tr>
<tr>
<td>Australian</td>
<td>138</td>
<td>0.001953</td>
<td>1</td>
<td>481.0</td>
<td>14.1</td>
</tr>
<tr>
<td></td>
<td>276</td>
<td>0.062500</td>
<td>1</td>
<td>1360.8</td>
<td>13.8</td>
</tr>
<tr>
<td></td>
<td>414</td>
<td>0.015625</td>
<td>16</td>
<td>3544.7</td>
<td>13.8</td>
</tr>
<tr>
<td>Diabetes</td>
<td>143</td>
<td>0.003906</td>
<td>4</td>
<td>561.3</td>
<td>31.4</td>
</tr>
<tr>
<td></td>
<td>286</td>
<td>0.125000</td>
<td>1</td>
<td>1480.1</td>
<td>24.0</td>
</tr>
<tr>
<td></td>
<td>429</td>
<td>0.062500</td>
<td>4</td>
<td>3762.3</td>
<td>23.0</td>
</tr>
<tr>
<td>Fourclass</td>
<td>172</td>
<td>0.062500</td>
<td>2</td>
<td>1479.2</td>
<td>28.9</td>
</tr>
<tr>
<td></td>
<td>344</td>
<td>4.000000</td>
<td>1</td>
<td>5493.5</td>
<td>7.2</td>
</tr>
<tr>
<td></td>
<td>516</td>
<td>4.000000</td>
<td>64</td>
<td>15952.8</td>
<td>0.2</td>
</tr>
<tr>
<td>German</td>
<td>200</td>
<td>0.125000</td>
<td>4</td>
<td>1099.1</td>
<td>27.3</td>
</tr>
<tr>
<td></td>
<td>400</td>
<td>0.003906</td>
<td>4</td>
<td>3540.2</td>
<td>25.8</td>
</tr>
<tr>
<td></td>
<td>600</td>
<td>0.007813</td>
<td>8</td>
<td>9824.7</td>
<td>24.7</td>
</tr>
<tr>
<td>Splice</td>
<td>200</td>
<td>0.500000</td>
<td>1</td>
<td>920.1</td>
<td>46.1</td>
</tr>
<tr>
<td></td>
<td>400</td>
<td>0.062500</td>
<td>1</td>
<td>3502.8</td>
<td>29.0</td>
</tr>
<tr>
<td></td>
<td>600</td>
<td>0.031250</td>
<td>2</td>
<td>10107.1</td>
<td>17.6</td>
</tr>
<tr>
<td>Titanic</td>
<td>440</td>
<td>0.031250</td>
<td>1</td>
<td>5509.0</td>
<td>21.8</td>
</tr>
<tr>
<td></td>
<td>880</td>
<td>0.015625</td>
<td>64</td>
<td>31215.1</td>
<td>21.8</td>
</tr>
<tr>
<td></td>
<td>1320</td>
<td>0.250000</td>
<td>32</td>
<td>118905.0</td>
<td>20.9</td>
</tr>
</tbody>
</table>

VI. CONCLUSION

In the core of SVM training lies a convex optimization problem which scales with $O(n^3)$. But in many real-world applications, it is prohibitive to directly train SVM for every candidate parameter.

In this paper, we have presented and analyzed an algorithm APT which provides an approximate result of parameter tuning. We use a low dimension and low rank approximate kernel matrix generated by MoCIC to replace the original matrix in the inner-layer of APT. Because of the special structure of the approximate kernel matrix generated by MoCIC, we can obtain the approximate optimal parameter much more efficiently than the accurate optimal parameter, while the TSA between different optimal models are very close.
matrix, parameter tuning algorithm could be conducted more efficiently. We make a time complexity comparison between the APT approximate algorithms and other parameter tuning algorithm. By experiments on 10 artificial and benchmark datasets, we verify the feasibility and the efficiency of APT. Experimental results show that the approximate parameter tuning approach is a sound and efficient parameter tuning framework.

ACKNOWLEDGMENT

The work was supported in part by the National Natural Science Foundation of China under grant No. 61170019 and the Natural Science Foundation of Tianjin under grant No. 11JCYBJC00700.

### REFERENCES


### TABLE III.
ACCURATE OPTIMAL MODELS AND APPROXIMATE OPTIMAL MODELS

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$(\gamma', C)_{\text{acc}}$</th>
<th>$(\gamma', C)_{\text{app}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sonar</td>
<td>(0.062500, 32)</td>
<td>(0.031250, 16)</td>
</tr>
<tr>
<td>Heart</td>
<td>(0.003906, 8)</td>
<td>(0.000061, 512)</td>
</tr>
<tr>
<td>Ionosphere</td>
<td>(0.062500, 4)</td>
<td>(0.500000, 2)</td>
</tr>
<tr>
<td>Breast</td>
<td>(0.031250, 32)</td>
<td>(0.007813, 1)</td>
</tr>
<tr>
<td>Australian</td>
<td>(0.250000, 1)</td>
<td>(0.062500, 1)</td>
</tr>
<tr>
<td>Diabetes</td>
<td>(0.125000, 4)</td>
<td>(0.015625; 64)</td>
</tr>
<tr>
<td>Fourclass</td>
<td>(2.000000, 256)</td>
<td>(4.000000, 1)</td>
</tr>
<tr>
<td>German</td>
<td>(0.01953, 32)</td>
<td>(0.003906, 4)</td>
</tr>
<tr>
<td>Splice</td>
<td>(0.007813, 16)</td>
<td>(0.062500, 1)</td>
</tr>
<tr>
<td>Titanic</td>
<td>(0.031250, 512)</td>
<td>(0.015625; 64)</td>
</tr>
</tbody>
</table>

### TABLE IV.
TIME AND TSA COMPARISON OF ACCURATE AND APPROXIMATE OPTIMAL MODELS

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Time (s)</th>
<th>Time (s)</th>
<th>TSA (%)</th>
<th>TSA (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sonar</td>
<td>522.2</td>
<td>132.9</td>
<td>87.1</td>
<td>86.2</td>
</tr>
<tr>
<td>Heart</td>
<td>1857.3</td>
<td>429.1</td>
<td>85.2</td>
<td>83.7</td>
</tr>
<tr>
<td>Ionosphere</td>
<td>2083.3</td>
<td>431.6</td>
<td>94.6</td>
<td>94.4</td>
</tr>
<tr>
<td>Breast</td>
<td>14423.5</td>
<td>2334.9</td>
<td>98.1</td>
<td>96.8</td>
</tr>
<tr>
<td>Australian</td>
<td>7690.4</td>
<td>1360.9</td>
<td>85.8</td>
<td>86.1</td>
</tr>
<tr>
<td>Diabetes</td>
<td>9784.4</td>
<td>1480.1</td>
<td>75.3</td>
<td>76.5</td>
</tr>
<tr>
<td>Fourclass</td>
<td>35839.8</td>
<td>5493.5</td>
<td>99.9</td>
<td>99.9</td>
</tr>
<tr>
<td>German</td>
<td>25902.0</td>
<td>3540.2</td>
<td>74.5</td>
<td>73.4</td>
</tr>
<tr>
<td>Splice</td>
<td>26961.2</td>
<td>3502.8</td>
<td>89.8</td>
<td>88.2</td>
</tr>
<tr>
<td>Titanic</td>
<td>27043.2</td>
<td>31215.1</td>
<td>77.6</td>
<td>78.1</td>
</tr>
</tbody>
</table>


Chenhao Yang was born in Henan province, 1988. Sep 2006 to Jul 2010: TianJin University, Computer Science and Technology professional bachelor. Sep 2010 to now: TianJin University, Computer Science and Technology academic master. Mr Yang’s main research interests stochastic and random algorithm for support vector machines.

Lingzhong Ding was born in Inner Mongolia province, 1986. Sep 2005 to Jul 2009: TianJin University, Computer Science and Technology professional bachelor. Sep 2009 to now: TianJin University, Computer Science and Technology PhD. Dr Ding’s main research interests include machine learning and model selection. He’s a student member of China Computer Federation.


He worked as professor in Liaoning Normal University from Jun 1988 to Jun 2003. From Jul 2003 until now, he is professor and PhD supervisor in TianJin University.

Prof Liao’s main research interests include artificial intelligence and theoretical computer science. Prof Liao is a member of China Computer Federation.