Quantum-Inspired Support Vector Machine

Chen Ding, Tian-Yi Bao, and He-Liang Huang*

Abstract—Support vector machine (SVM) is a particularly powerful and flexible supervised learning model that analyzes data for both classification and regression, whose usual algorithm complexity scales polynomially with the dimension of data space and the number of data points. To tackle the big data challenge, a quantum SVM algorithm was proposed, which is claimed to achieve exponential speedup for least squares SVM (LS-SVM). Here, inspired by the quantum SVM algorithm, we present a quantum-inspired classical algorithm for LS-SVM. In our approach, an improved fast sampling technique, namely indirect sampling, is proposed for sampling the kernel matrix and classifying. We first consider the LS-SVM with a linear kernel, and then discuss the generalization of our method to non-linear kernels. Theoretical analysis shows our algorithm can make classification with arbitrary success probability in logarithmic runtime of both the dimension of data space and the number of data points for low rank, low condition number and high dimensional data matrix, matching the runtime of the quantum SVM.

Index Terms—Quantum-inspired algorithm, machine learning, support vector machine, exponential speedup, matrix sampling.

I. INTRODUCTION

SINCE the 1980s, quantum computing has attracted wide attention due to its enormous advantages in solving hard computational problems [1], such as integer factorization [2–4], database searching [5, 6], machine learning [7–11] and so on [12, 13]. In 1997, Daniel R. Simon offered compelling evidence that the quantum model may have significantly more complexity theoretic power than the probabilistic Turing machine [14]. However, it remains an interesting question where is the border between classical computing and quantum computing. Although many proposed quantum algorithms have exponential speedups over the existing classical algorithms, is there any way we can accelerate such classical algorithms to the same complexity of the quantum ones?

In 2018, inspired by the quantum recommendation system algorithm proposed by Iordanis Kerenidis and Anupam Prakash [15], Ewin Tang designed a classical algorithm to produce a recommendation algorithm that can achieve an exponential improvement on previous algorithms [16], which is a breakthrough that shows how to apply the subsampling strategy based on Alan Frieze, Ravi Kannan, and Santosh Vempala’s 2004 algorithm [17] to find a low-rank approximation of a matrix. Subsequently, Tang continued to use same techniques to dequantize two quantum machine learning algorithms, quantum principal component analysis [18] and quantum supervised clustering [19], and shows classical algorithms could also match the bounds and runtime of the corresponding quantum algorithms, with only polynomial slowdown [20].

Later, András Gilyén et al. [21] and Nai-Hui Chia et al. [22] independently and simultaneously proposed a quantum-inspired matrix inverse algorithm with logarithmic complexity of matrix size, which eliminates the speedup advantage of the famous Harrow-Hassidim-Lloyd (HHL) algorithm [23] on certain conditions. Recently, Juan Miguel Arrazola et al. studied the actual performance of quantum-inspired algorithms and found that quantum-inspired algorithms can perform well in practice under given conditions. However, the conditions should be further reduced if we want to apply the algorithms to practical datasets [24]. All of these works give a very promising future for designing the quantum-inspired algorithm in the machine learning area, where matrix inverse algorithms are universally used.

Support vector machine (SVM) is a data classification algorithm which is commonly used in machine learning area [25, 26]. Extensive studies have been conducted on SVMs to boost and optimize their performance, such as the sequential minimal optimization algorithm [27], the cascade SVM algorithm [28], and the SVM algorithms based on Markov sampling [29, 30]. These algorithms offer promising speedup either by changing the way of training a classifier, or by reducing the size of training sets. However, the time complexity of current SVM algorithms are all polynomial of data sizes. In 2014, Patrick Rebentrost, Masoud Mohseni and Seth Lloyd proposed the quantum SVM algorithm [31], which can achieve an exponential speedup compared to the classical SVMs. The time complexity of quantum SVM algorithm is polynomial of the logarithm of data sizes. Inspired by the quantum SVM algorithm, Tang’s methods [16] and András Gilyén et al.’s work [21], we propose a quantum-inspired classical SVM algorithm, which also shows exponential speedup compared to previous classical SVM for low rank, low condition number and high dimensional data matrix. Both quantum SVM algorithm [31] and our quantum-inspired SVM algorithm are least squares SVM (LS-SVM), which reduce the optimization problem to finding the solution of a set of linear equations.

Our algorithm is a dequantization of the quantum SVM algorithm [31]. In quantum SVM algorithm, the labeled data vectors \( x_j \) for \( j = 1, \ldots, m \) are mapped to quantum vectors \( |x_j\rangle = 1/|x_j| \sum(x_j)_{k} |k\rangle \) via a quantum random access...
memory (qRAM) and the kernel matrix is prepared using quantum inner product evaluation [19]. Then the solution of SVM is found by solving a linear equation system related to the quadratic programming problem of SVM using the quantum matrix inversion algorithm [23]. In our quantum-inspired SVM, the labeled vectors are stored in an arborescent data structure which provides the ability to random sampling within logarithmic time of the vector lengths. By performing sampling on these labeled vectors both by their numbers and lengths to get a much smaller dataset, we then find the approximate singular value decomposition of the kernel matrix. And finally, we solve the optimization problem and perform classification based on the solved parameters.

Our methods, particularly the sampling technique, is based on [16], [21]. However, the previous sampling techniques cannot be simply copied to solve the SVM tasks, since we don’t have an efficient direct sampling access to the kernel matrix we want to perform matrix inversion on (see Section II-B for a more detailed explanation). Hence we have developed an indirect sampling technique to solve such problem. In the whole process, we need to avoid the direct multiplication on the vectors or matrices with the same size as the kernel, in case losing the exponential speedup. We first consider the LS-SVM with linear kernels, no regularization terms and no bias of the classification hyperplane, which could be regarded as the prototype for quantum-inspired techniques applied in various SVMs. Then we show that the regularization terms can be easily included in the algorithm in Section [III]. Finally, we discuss the generalization of our method to non-linear kernels in Section [VII-C] and the general case without the constraint on biases of classification hyperplanes in Section [VII-D]. Theoretical analysis shows that our quantum-inspired SVM can achieve exponential speedup over existing classical algorithms under several conditions. Experiments are carried out to demonstrate the feasibility of our algorithm. The indirect sampling developed in our work opens up the possibility of a wider application of the sampling methods into the field of machine learning.

II. PRELIMINARY

A. Notations

We list some matrix-related notations used in this paper.

B. Least squares SVM

Suppose we have m data points \{(x_j, y_j) : x_j \in \mathbb{R}^n, y_j = \pm 1 \}_{j=1}^{m}, where y_j is \pm 1 depending on the class which x_j belongs to. Denote (x_1, \ldots, x_m) by X and (y_1, \ldots, y_m)^T by y. A SVM finds a pair of parallel hyperplanes \( x \cdot w + b = \pm 1 \) that divides the points into two classes depending on the given data. Then for any new input points, it can make classification by its relative position with the hyperplanes.

We make the following assumption on the dataset so as to simplify the problem: Assume these data points are equally distributed on both sides of a hyperplane that passes through the origin and their labels are divided by such hyperplane. Thus we assume \( b = 0 \). An generalized method for \( b \neq 0 \) is discussed in Section [VII-D].

A. Least squares SVM

I. PRELIMINARY

A. Notations

We list some matrix-related notations used in this paper.

B. Least squares SVM

Suppose we have m data points \{(x_j, y_j) : x_j \in \mathbb{R}^n, y_j = \pm 1 \}_{j=1}^{m}, where y_j is \pm 1 depending on the class which x_j belongs to. Denote (x_1, \ldots, x_m) by X and (y_1, \ldots, y_m)^T by y. A SVM finds a pair of parallel hyperplanes \( x \cdot w + b = \pm 1 \) that divides the points into two classes depending on the given data. Then for any new input points, it can make classification by its relative position with the hyperplanes.

We make the following assumption on the dataset so as to simplify the problem: Assume these data points are equally distributed on both sides of a hyperplane that passes through the origin and their labels are divided by such hyperplane. Thus we assume \( b = 0 \). An generalized method for \( b \neq 0 \) is discussed in Section [VII-D].

A. Notations

We list some matrix-related notations used in this paper.

B. Least squares SVM

Suppose we have m data points \{(x_j, y_j) : x_j \in \mathbb{R}^n, y_j = \pm 1 \}_{j=1}^{m}, where y_j is \pm 1 depending on the class which x_j belongs to. Denote (x_1, \ldots, x_m) by X and (y_1, \ldots, y_m)^T by y. A SVM finds a pair of parallel hyperplanes \( x \cdot w + b = \pm 1 \) that divides the points into two classes depending on the given data. Then for any new input points, it can make classification by its relative position with the hyperplanes.

We make the following assumption on the dataset so as to simplify the problem: Assume these data points are equally distributed on both sides of a hyperplane that passes through the origin and their labels are divided by such hyperplane. Thus we assume \( b = 0 \). An generalized method for \( b \neq 0 \) is discussed in Section [VII-D].

A. Notations

We list some matrix-related notations used in this paper.

B. Least squares SVM

Suppose we have m data points \{(x_j, y_j) : x_j \in \mathbb{R}^n, y_j = \pm 1 \}_{j=1}^{m}, where y_j is \pm 1 depending on the class which x_j belongs to. Denote (x_1, \ldots, x_m) by X and (y_1, \ldots, y_m)^T by y. A SVM finds a pair of parallel hyperplanes \( x \cdot w + b = \pm 1 \) that divides the points into two classes depending on the given data. Then for any new input points, it can make classification by its relative position with the hyperplanes.

We make the following assumption on the dataset so as to simplify the problem: Assume these data points are equally distributed on both sides of a hyperplane that passes through the origin and their labels are divided by such hyperplane. Thus we assume \( b = 0 \). An generalized method for \( b \neq 0 \) is discussed in Section [VII-D].

TABLE I

The Notations

| Symbol | Meaning |
|--------|---------|
| A      | matrix A |
| y      | vector y or matrix y with only one column |
| A^T   | transpose of A |
| A^+   | pseudo inverse of A |
| A^T_F | transpose of pseudo inverse of A |
| i      | i-th row of A |
| f     | f-th column of A |
| ∥A∥   | 2-operator norm of A |
| ∥A∥_F | Frobenius norm of A |
| Q(·)  | time complexity for querying an element of · |
| L(·)  | time complexity for sampling an element of · |

According to [20], the optimization problem of LS-SVM with linear kernel is

\[
\min_{w,b,e} \mathcal{L}_1(w,b,e) = \frac{1}{2} w^T w + \frac{\gamma}{2} \sum_{k=1}^{m} e_k^2,
\]

subject to \( y_k (w^T x_k + b) = 1 - e_k, \quad k = 1, \ldots, m \).

Take \( b = 0 \), we get

\[
\min_{w,e} \mathcal{L}_2(w,e) = \frac{1}{2} w^T w + \frac{\gamma}{2} \sum_{k=1}^{m} e_k^2,
\]

subject to \( y_k w^T x_k = 1 - e_k, \quad k = 1, \ldots, m \).

One defines the Lagrangian

\[
\mathcal{L}(w,e,\mu) = \mathcal{L}_2(w,e) - \sum_{k=1}^{m} \mu_k (y_k w^T x_k - 1 + e_k).
\]

The condition for optimality

\[
\frac{\partial \mathcal{L}}{\partial w} = 0 \Rightarrow w = \sum_{k=1}^{m} \mu_k y_k x_k,
\]

\[
\frac{\partial \mathcal{L}}{\partial e_k} = 0 \Rightarrow \mu_k = \gamma e_k, \quad k = 1, \ldots, m,
\]

\[
\frac{\partial \mathcal{L}}{\partial \mu_k} = 0 \Rightarrow y_k w^T x_k - 1 + e_k = 0, \quad k = 1, \ldots, m.
\]

can be written as the solution to the following set of linear equations \( Z^T Z \mu + \gamma^{-1} \mu = 1 \), where \( Z = (x_1y_1, \ldots, x_my_m) \). Let \( \alpha_k = \mu_k y_k \), we have

\[
(X^T X + \gamma^{-1} I) \alpha = y. \tag{1}
\]

Once \( \alpha \) is solved, the classification hyperplane will be \( x^T X \alpha = 0 \). Given query point \( x \), we evaluate \( \text{sgn}(x^T X \alpha) \) to make classification.

We use our sampling techniques in solving Equation (1) and evaluating \( \text{sgn}(x^T X \alpha) \) to avoid time complexity overhead of poly(m) or poly(n), which will kill the wanted exponential speedup. Note that the quantum-inspired algorithm for linear equations [21], [22] may inverse a low-rank matrix in logarithmic runtime. However, such algorithm cannot be invoked directly to solve Equation (1) here, since the complexity of direct computing the matrix \( X^T X + \gamma^{-1} I \) is polynomial, which would once again kill the exponential speedup. Thus we

We make the following assumption on the dataset so as to simplify the problem: Assume these data points are equally distributed on both sides of a hyperplane that passes through the origin and their labels are divided by such hyperplane. Thus we assume \( b = 0 \). An generalized method for \( b \neq 0 \) is discussed in Section [VII-D].
need to develop the indirect sampling technique to efficiently perform matrix inversion on $X^T X + \gamma^{-1} I$ with only sampling access of $X$.

C. The sampling technique

We show the definition and idea of our sampling method to get indices, elements or submatrices, which is the key technique used in our algorithm, as well as in [16, 17, 21].

Definition 1 (Sampling on vectors). Suppose $v \in \mathbb{C}^n$, define $q^{(v)}$ as a probability distribution that:

$$x \sim q^{(v)} : \quad P[x = i] = \frac{|v_i|^2}{||v||^2}.$$ 

Picking an index according to the probability distribution $q^{(v)}$ is called a sampling on $v$.

Definition 2 (Sampling the indices from matrices). Suppose $A \in \mathbb{C}^{m \times n}$, define $q^{(A)}$ as a 2-dimensional probability distribution that:

$$(x, y) \sim q^{(A)} : \quad P[x = i, y = j] = \frac{|A_{ij}|^2}{||A||_F^2}.$$ 

Picking a pair of indices $(i, j)$ according to the probability distribution $q^{(A)}$ is called a sampling on $A$.

Definition 3 (Sampling the submatrices from matrices). Suppose the target is to sample a submatrix $X'' \in \mathbb{C}^{c \times r}$ from $X \in \mathbb{C}^{m \times n}$. First we sample $r$ times on the vector $(|X_{i,j}|)_{j=1,\ldots,m}$ and get column indices $j_1, \ldots, j_r$. The columns $X_{c,j_1}, \ldots, X_{c,j_r}$ form submatrix $X'$. Then we sample $c$ times on the $j$-th column of $X$ and get row indices $i_1, \ldots, i_c$. In each time the $j$ is sampled uniformly at random from $j_1, \ldots, j_r$. The rows $X'_{i_1, \ldots, i_c}$, $X'_{i_1, \ldots, i_c}$ form submatrix $X''$. The matrices $X'$ and $X''$ are normalized so that $\mathbb{E}[X'X'^T] = XX^T$ and $\mathbb{E}[X'^T X''] = X''^T X''$.

The process of sampling the submatrices from matrices (as described in Def. 3) is shown in Fig. 1. To put it simple, it is taking several rows and columns out of the matrix by a random choice decided by the “importance” of the elements. Then normalize them so that they are unbiased from the original rows and columns.

To achieve fast sampling, we usually store vectors in an arboreose data structure (such as binary search tree) as suggested in [16] and store matrices by a list of their row trees or column trees. Actually, the sampling is an analog of quantum states measurements. It only reveals a low-dimensional projection of vectors and matrices in each calculation. Rather than computing with the whole vector or matrix, we choose the most representative elements for them with a high probability (we choose the elements according to the probability of their squares, which is also similar to the quantum measurement of quantum states.). The sampling technique we use has the advantage of unbiasedly representing the original vector while consuming less computing resources.

We note that there are other kinds of sampling methods for SVM such as the Markov sampling [29, 30]. Different sampling methods may work well on different scenarios. Our algorithm is designed for low-rank datasets, while the algorithms based on Markov sampling [29, 30] may work well on the datasets that the columns form a uniformly ergodic Markov chain. In our algorithm, to achieve exponential speedup, the sampling technique is different from Markov sampling: (i) We sample both the rows and columns of matrix, rather than only sampling columns. (ii) We sample each elements according to norm-squared probability distribution. (iii) In each dot product calculation (Alg. 1), we use sampling technique to avoid operations with high complexity.

D. The preliminary algorithms

We invoke two algorithms employing sampling techniques for saving complexity from [21]. They are treated as oracles that outputs certain outcomes with controlled errors in the main algorithm. Lemma 3 and Lemma 4 shows their correctness and efficiency. For the sake of convenience, some minor changes on the algorithms and lemmas are made.

1) Trace inner product estimation: Alg. 1 achieves calculation of trace inner products with logarithmic time on the sizes of the matrices.

Algorithm 1 Trace Inner Product Estimation.

Input: $A \in \mathbb{C}^{m \times n}$ that we have sampling access in complexity $L(A)$ and $B \in \mathbb{C}^{n \times m}$ that we have query access in complexity $Q(B)$. Relative error bound $\xi$ and success probability bound $1 - \eta$.

Goal: Estimate $\text{Tr}[AB]$.

1: Repeat step 2 $\left[\frac{\log(1/\eta)}{\xi^2}\right]$ times and take the median of $Y$, noted as $Z$.
2: Repeat step 3 $\left[\frac{9}{\xi^2}\right]$ times and calculate the mean of $X$, noted as $Y$.
3: Sample $i$ from row norms of $A$. Sample $j$ from $A_i$. Let $X = \frac{1}{\|A_i\|_F^2} A_i B_{ji}$.

Output: $Z$.

Lemma 3 [21]. Suppose that we have length-square sampling access to $A \in \mathbb{C}^{m \times n}$ and query access to the matrix $B \in \mathbb{C}^{n \times m}$ in complexity $Q(B)$. Then we can estimate $\text{Tr}[AB]$ to precision $\xi\|A\|_F\|B\|_F$ with probability at least $1 - \eta$ in time $O\left(\frac{\log(1/\eta)}{\xi^2}(L(A) + Q(B))\right)$.

Algorithm 2 Rejection sampling.

Input: $A \in \mathbb{C}^{m \times n}$ that we have length-square sampling access and $b \in \mathbb{C}^n$ that we have norm access and $y = Ab$ that we have query access.

Goal: Sample from length-square distribution of $y = Ab$.

1: Take $D \geq \|b\|_2^2$.
2: Sample a row index $i$ by row norm squares of $A$.
3: Query $|y_i|^2 = |A_{i,b}|^2$ and calculate $\frac{|A_{i,b}|^2}{D |A_{i,b}|^2}$.
4: Sample a real number $x$ uniformly distributed in $[0, 1]$. If $x < \frac{|A_{i,b}|^2}{D |A_{i,b}|^2}$, output $i$, else, go to step 2.

Output: The row index $i$. 
2) **Rejection sampling**: Alg. 2 achieves sampling of a vector that we do not have full query access in time logarithmic of its length.

**Lemma 2 [21]**. Suppose that we have length-square sampling access to \( A \in \mathbb{C}^{m \times n} \) having normalized rows, and we are given \( b \in \mathbb{C}^n \). Then we can implement queries to the vector \( y := A b \in \mathbb{C}^n \) with complexity \( \mathcal{Q}(y) = O(nQ(A)) \) and we can length-square sample from \( q(y) \) with complexity \( \mathcal{L}(y) \) such that 

\[
\mathbb{E}[L(y)] = O\left(\frac{n|b|^2}{\mathbb{E}y^2}\right) (L(A) + nQ(A)).
\]

**III. QUANTUM-INSPIRED SVM ALGORITHM**

We show the main algorithm (Alg. 3) that makes classification as the classical SVMs do. Note that actual calculation only happens when we use the expression "calculate" in this algorithm. Otherwise it will lose the exponential-speedup advantage for operations on large vectors or matrices. \( y \) is temporarily taken as \( \infty \). Fig. 2 shows the algorithm process.

**Algorithm 3 Quantum-inspired SVM Algorithm.**

**Input**: \( m \) training data points and their labels \((x_j, y_j) : x_j \in \mathbb{R}^n, y_j = \pm 1 \}_{j=1,...,m}\), where \( y_j = \pm 1 \) depending on the class to which \( x_j \) belongs. Error bound \( \epsilon \) and success probability bound \( 1 - \eta \), \( \gamma \) set as \( \infty \).

**Goal 1**: Find \( \tilde{\alpha} \) that \( \|\tilde{\alpha} - \alpha\| \leq \epsilon \|\alpha\| \) with success probability at least \( 1 - \eta \), in which \( \alpha = (X^T X)^y \).

**Goal 2**: For any given \( x \in \mathbb{R}^n \), find its class.

1. **Init**: Set \( r, c \) as described in [6] and [7].
2. **Sample columns**: Sample \( r \) column indices \( i_1, i_2, ..., i_r \) according to the column norm squares \( \|x_{:,j}\|^2 \). Define \( X' \) to be the matrix whose \( s \)-th column is \( \frac{\|x_{:,j}\|^2}{\mathbb{E}x_{:,j}^2} x_{:,j} \). Define \( A' = X'^T X' \).

**Algorithm 3 Quantum-inspired SVM Algorithm.**

3. **Sample rows**: Sample \( s \in [r] \) uniformly, then sample a row index \( j \) distributed as \( \frac{|x_{:,j}|^2}{\mathbb{E}x_{:,j}^2} \). Sample a total number of \( c \) row indices \( j_1, j_2, ..., j_c \) this way. Define \( X'' \) whose \( t \)-th row is \( \frac{\|x_{:,j}\|^2}{\mathbb{E}x_{:,j}^2} \). Define \( A'' = X''^T X'' \).

4. **Spectral decomposition**: Calculate the spectral decomposition of \( A'' \). Denote here by \( A'' = \sum_{l=1}^{\epsilon} \lambda_l V_l \). Let \( A'' \) be calculated eigenvalues by \( \frac{1}{2} \alpha \|e\|^2 \) for \( l = 1, ..., k \). Define \( A'' = X''^T X'' \).

5. **Approximate eigenvectors**: Let \( R = X''^T X. \) Define \( V_l = \frac{R V_l}{\sqrt{\lambda_l}} \). We retain \( k \) eigenvalues from which we conclude the time complexity.

6. **Estimate matrix elements**: Calculate \( \tilde{\alpha}_l = V_l^T y \) to precision \( \frac{\epsilon}{16\sqrt{n}} \|y\| \) by Alg. 1 each with success probability \( 1 - \frac{\eta}{\gamma} \). Let \( u = \sum_{l=1}^{\epsilon} \frac{\tilde{\alpha}_l}{\sqrt{\lambda_l}} V_l \).

7. **Find query access**: Find query access of \( \tilde{\alpha} = R^T u \) by \( \tilde{\alpha}_p = u^T R \tilde{a}_p \), in which \( R \tilde{a}_p \) is calculated to precision \( \frac{\epsilon}{2\sqrt{n}} \) by Alg. 1 each with success probability \( 1 - \frac{\eta}{\gamma} \).

8. **Find sign**: Calculate \( x^T X \tilde{\alpha} \) to precision \( \frac{\epsilon}{2} \|\alpha\| \|x\| \) with success probability \( 1 - \frac{\eta}{\gamma} \) by Alg. 1. Tell its sign.

**Output**: The answer class depends on the sign. Positive corresponds to 1 while negative to -1.

The following theorem states the accuracy and time complexity of quantum-inspired support vector machine algorithm, from which we conclude the time complexity \( T \) depends polynomially on \( m, n \) and polynomially on \( k, \epsilon, \eta, \gamma \). It is to be proved in section [IV] and section [V].

**Theorem 1**: Given parameters \( \epsilon > 0, 0 < \eta < 1 \), and given the data matrix \( X \) with size \( m \times n \), rank \( k \), norm \( 1 \), and condition number \( \kappa \), the quantum-inspired SVM algorithm will find the classification expression \( x^T X \alpha \) for any vector \( x \in \mathbb{C}^n \) with error less than \( \epsilon \kappa^2 \sqrt{m} \|x\| \), success probability higher than
1 − η and time complexity $T(m, n, k, κ, ε, η)$.

$$T = O(r \log_2 m + cr \log_2 n + r^3$$

$$+ \frac{\|X\|^2}{ε^2} \log_2(\frac{8k}{η})(\log_2(mn) + k)$$

$$+ \frac{1}{ε^2} \log_2 1 + \log_2(mn) + r \log_2 \left( \frac{2}{η_1} \frac{\|X\|^4}{ε^2 r} \log_2(mn) \right),$$

in which

$$ε_1 = \frac{ε\|x\|}{2\sqrt{\frac{16}{e^2}}} \log_2 \left( \frac{16}{ε^2} \right),$$

$$η_1 = \frac{ε\|x\|}{8\sqrt{\frac{16}{e^2}}} \log_2 \left( \frac{16}{ε^2} \right).$$

In Alg. 3, $γ$ is set as $∞$, which makes the coefficient matrix $A = X^TX$. Notice that the eigenvectors of $X^TX + γ^−1I$ and $X^TX$ are the same, and the difference of their eigenvalues are $γ^−1$. Thus the algorithm can be easily extended to be applied to the coefficient matrix $X^TX + γ^−1I$ with arbitrary $γ$, by just simply adding $γ^−1$ to the calculated eigenvalues in Step 4.

**IV. ACCURACY**

We prove that the error of computing the classification expression $x^TX\tilde{α}$ in the quantum-inspired SVM algorithm will not exceed $εk^2 \sqrt{m}\|x\|$. We take $γ = ∞$ in the analysis because adding $γ^−1$ to the eigenvalues won’t cause error and thus the analysis is the same in the case of $γ ≠ ∞$. We first show how to break the total error into multiple parts, and then analyze each part in the subsections.

Let $α = (X^TX)^+y$, $α' = \sum_{l=1}^k \frac{λ_l}{σ_l^2} \tilde{V}_l = \tilde{V}\Sigma^{-2}Y^TY$ in which $λ_l = \tilde{V}_l^TY$ and $α'' = \sum_{l=1}^k \frac{1}{σ_l^2} \tilde{V}_l$. Then the total error of the classification expression is $E$.

$$E = Δ(x^TXα)$$

$$≤ \|x^T(α - \tilde{α})\| + Δ(x^TX\tilde{α})$$

$$≤ \|x\|(\|α - α'\| + \|α' - α''\| + \|α'' - \tilde{α}\|) + Δ(x^TX\tilde{α})$$

Denote $E_1 = \|x\|\|α - α'\|$, $E_2 = \|x\|\|α' - α''\|$, $E_3 = \|x\|\|α'' - \tilde{α}\|$, $E_4 = Δ(x^TX\tilde{α})$. Our target is to show each of them is no more than $\frac{ε}{4}\|α\|\|x\|$ with probability no less than $1 - \frac{η}{4}$.

So that

$$E ≤ E_1 + E_2 + E_3 + E_4$$

$$≤ εk^2 \sqrt{m}\|x\|,$$

with success probability no less than $1 - η$.

$E_1$ represents the error introduced by subsampling and eigenvector approximation (i.e., Step 1-5 in Alg. 3). The fact that it is less than $\frac{ε}{4}\|α\|\|x\|$ with probability no less than $1 - \frac{η}{4}$ is shown in subsection IV-B

$E_2$ represents the error introduced by approximation on $λ_l$ (i.e., Step 6 in Alg. 3). The fact that it is less than $\frac{ε}{4}\|x\|\|x\|$ with probability no less than $1 - \frac{η}{4}$ is shown in subsection IV-A.

$E_3$ represents the error introduced in query of $R$ and $α$. The fact that it is less than $\frac{ε}{4}\|α\|\|x\|$ with probability no less than $1 - \frac{η}{4}$ is guaranteed by Step 7 of Alg. 3.

1 For any expression $f$, $Δ(f)$ represents the difference of the exact value of $f$ and the value calculated by the estimation algorithms Alg. 1 and Alg. 2 (These two algorithms cannot get the exact values because randomness is introduced).
When a submatrix $X''$ is randomly subsampled from $X'$, it is a matrix of multiple random variables. Theorem 2 is the Chebyshev’s Inequality for $X''$. It points out that the operator norm distance between $X'' X'$ and $X'''' X''''$ is short with a high probability.

**Theorem 3.** Suppose the columns of matrix $V''$, denoted as $V'_l, l = 1, \ldots, k$, are orthogonal normalized vectors while

$$
A'' = \sum_{l=1}^{k} \sigma_l^2 V'_l V'_l^T.
$$

Suppose $\|A' - A''\| \leq \beta$. Then $\forall i, j \in \{1, ..., r\}$,

$$
|V''_i^T A' V'_j - \delta_{ij} \sigma_l^2| \leq \beta.
$$

Theorem 3 points out that if matrix $A'$ and $A''$ are close in operator norm sense, $A''$’s eigenvectors will approximately work as eigenvectors for $A'$ too.

**Theorem 4.** Suppose the columns of matrix $V''$, denoted as $V'_l, l = 1, \ldots, k$, are orthogonal normalized vectors while

$$
|V''_i^T A' V'_j - \delta_{ij} \sigma_l^2| \leq \beta, \quad \forall i, j \in \{1, ..., r\}.
$$

\[\text{Suppose } ||X X^T - X' X''|| \leq \epsilon', ||X|| \leq 1, \frac{1}{\sigma_i^2} \leq \epsilon' \leq 1 \text{ and the condition of Theorem 3 satisfies. Let } \tilde{V}_i = \frac{V_i^T}{\sigma_i}, \text{ then }\]

$$
|\tilde{V}_i^T \tilde{V}_j - \delta_{ij}| \leq \epsilon'' + \beta ||X||^2, \quad \text{and }\]

$$
|\tilde{V}_i^T A \tilde{V}_j - \delta_{ij} \sigma_l^2| \leq (2\epsilon'' + \beta ||X||^2) \sigma_l^2.
$$

in which $A' = X'X$, $A = X^T X$.

Theorem 4 points out that if $A''$’s eigenvectors approximately work as eigenvectors for $A'$ and $||X X^T - X' X''|| \leq \epsilon'$, $V''_l$ approximately work as eigenvectors for $A$.

**Theorem 5** [21]. If rank($B$) $\leq k$, $V$ has $k$ columns that spans the row and column space of $B$, then

$$
||B|| \leq ||(V^T V)^+|| ||V^T B V||. \quad (3)
$$

Under the condition that $\tilde{V}_l$ approximately work as eigenvectors for $A$, the following Theorem 6 points out that $V^T V''$ is functionally close to the inverse of matrix $A$.

**Theorem 6.** If $\forall i, j \in \{1, ..., k\}$,

$$
|\tilde{V}_i^T \tilde{V}_j - \delta_{ij}| \leq \frac{1}{4k}, \quad (2)
$$

$$
|\tilde{V}_i^T A \tilde{V}_j - \delta_{ij} \sigma_l^2| \leq \zeta, \quad \text{and the condition of Theorem 4 satisfies. Then }\]

$$
||\tilde{V}^T V^T A - I_m|| \leq \frac{5}{3} k^2 \zeta. \quad (3)
$$

To conclude, for $\mathbb{P}(|\alpha' - \alpha| > \frac{\epsilon}{4})$, we need to pick $\epsilon'$ and $\beta$ such that

$$
k^2 \beta^2 + 2k \beta + k^2 \epsilon' ||X||^2 \leq \frac{1}{4k}. \quad (3)
$$
\[(2\epsilon + \beta \|X\|_F^2)\|X\|_F^2 k^2 \leq \zeta,\]
\[
\frac{5}{3} kk \zeta \leq \frac{\epsilon}{4},
\]
and decide the sampling parameter as
\[
r = \left\lfloor \frac{4 \log_2(\frac{2\alpha}{\eta})}{\epsilon^2} \right\rfloor,
\]
\[
c = \left\lfloor \frac{4\epsilon^2 \log_2(\frac{\alpha}{\eta})}{\beta^2} \right\rfloor.
\]

**B. Proof of \( E_1 \leq \frac{\epsilon}{4} \|\alpha\| \|x\| \)**

Notice that
\[
E_4 = \|x\| \|\alpha - \tilde{\alpha}\|.
\]
For \( y = X^T \alpha \) and \( \alpha = X^+ X^T \gamma \), we have \( \|y\| \leq \|\alpha\| \leq k^2 \|\gamma\| \).

For \( \|\tilde{\alpha} - \alpha'\| \), let \( z \) be the vector that \( z_i = \frac{\alpha_i - \tilde{\alpha}_i}{\sigma_i^2} \), we have
\[
\|\tilde{\alpha} - \alpha'\| = \| \sum_{i=1}^k \frac{\alpha_i - \tilde{\alpha}_i}{\sigma_i^2} \tilde{v}_i \| = \| \tilde{v}_z \| \leq \sqrt{\|\tilde{v}_z\|^2 \|z\|^2} \leq \frac{4}{3} \frac{3\epsilon^2 \sigma_i^2}{8k} \|y\| \frac{1}{\sigma_i^2} \sqrt{k} \leq \frac{1}{4} \epsilon \|\alpha\|,
\]
in which \( \|\tilde{v}_z\| \leq \frac{4}{3} \) as shown in proof of theorem 6.

**V. COMPLEXITY**

In this section, we will analyze the time complexity of each step in the main algorithm. We divide these steps into four parts and analyze each part in each subsection: Step 1-3 are considered in Subsection V-A. Step 4 is considered in Subsection V-B. Step 5-6 are considered in Subsection V-C. Step 7-8 are considered in Subsection V-D. Note that in the main algorithm the variables \( R, \tilde{V}, \tilde{\alpha} \) are queried instead of calculated. We include the corresponding query complexity in analysis of the steps where we queried these variables.

**A. Sampling of columns and rows**

In Step 1, the value of \( r \) and \( c \) are determined according to Inequalities (3.5)-(3.7). The time of solving these inequalities is a constant. In Step 2 we sample \( r \) indices, each sampling takes no more than \( \log_2 m \) time according to the arborescent vector data structure shown in II-C. In Step 3 we sample \( c \) indices, each sampling takes no more than \( r \log_2 n \) time according to the arborescent matrix data structure shown in II-C. Thus the overall time complexity of Step 1-3 is \( O(r \log_2 m + cr \log_2 n) \).

**B. The spectral decomposition**

Step 4 is the spectral decomposition. For \( r \times r \) symmetric matrix \( A \), the fastest classical spectral decomposition is through classical spectral symmetric QR method, of which the complexity is \( O(r^2) \).

**C. Calculation of \( \tilde{\lambda} \)**

In Step 5-6 we calculate \( \tilde{\lambda} \). By Alg. [1], we have
\[
\lambda_i = \frac{1}{\sigma_i^2} \tilde{V}_i^T R \gamma = \frac{1}{\sigma_i^2} \text{Tr} \{ \tilde{V}_i^T X^T \gamma \} = \frac{1}{\sigma_i^2} \text{Tr} \{ X \gamma \tilde{V}_i^T X^T \}.
\]

Observe that \( \| \gamma \tilde{V}_i^T X^T \|_F = \| \gamma \| \| \tilde{V}_i^T X^T \| \leq \| \gamma \| \), and we can query the \((i, j)\) matrix element of \( \gamma \tilde{V}_i^T X^T \) in cost \( O(r) \). According to Lemma 1 the complexity in step 6 is
\[
T_6 = O\left( \frac{\|X\|_F^2 \log_2 (\frac{8k}{\eta})}{\epsilon^2} \text{log}(\text{log}_2(mn) + k) \right).
\]

**D. Calculation of \( x^T X \tilde{\alpha} \)**

In Step 7-8 we calculate \( x^T X \tilde{\alpha} \). Calculation of \( x^T X \tilde{\alpha} \) is the last step of the algorithm, and also the most important step for saving time complexity. In Step 8 of Alg. 1, we need to calculate \( x^T X \tilde{\alpha} \), which is equal to \( \text{Tr} \{ X \tilde{\alpha} x^T \} \), with precision \( \epsilon \|\alpha\| \|x\| \) and success probability \( 1 - \frac{\eta}{8} \) using Alg. 1. Let the \( A \) and \( B \) in Alg. 1 be \( X \) and \( \tilde{\alpha} x^T \), respectively. To calculate \( \text{Tr} \{ X \tilde{\alpha} x^T \} \), we first establish the query access for \( \tilde{\alpha} x^T \) (we already have the sampling access of \( X \)), and then using the Alg. 1 as an oracle. We first analyze the time complexity of querying \( R \) and \( \tilde{\alpha} \), and then provide the time complexity of calculating \( x^T X \tilde{\alpha} \):

1) Query of \( R \): First we find query access of \( R = X^T X \). For any \( s = 1, \ldots, r, \ j = 1, \ldots, m, \ R_{sj} = e_s^T X e_j^T X e_j = \text{Tr} \{ X e_j^T X e_j e_s^T \} \), we calculate such trace by Alg. 1 to precision \( \epsilon_1 \) with success probability \( 1 - \frac{\eta}{8} \). The time complexity for one query will be
\[
Q(R) = O(\log_2 (\frac{2}{\eta_1}) \|X\|_F^4 \|x\| \log_2(mn)).
\]

2) Query of \( \tilde{\alpha} \): For any \( i = 1, \ldots, m \), we have \( \tilde{\alpha}_i = \sum_{s=1}^r R_{sj} u_s \). One query of \( \tilde{\alpha} \) will cost time \( r k Q(R) \), with error \( \epsilon_1 \sum_{s=1}^r |u_s| \) and success probability more than \( 1 - \frac{\eta}{8} \).

3) Calculation of \( x^T X \tilde{\alpha} \): We use Alg. 1 to calculate \( x^T X \tilde{\alpha} = \text{Tr} \{ X \tilde{\alpha} x^T \} \) to precision \( \frac{\epsilon}{4} \|\alpha\| \|x\| \) with success probability \( 1 - \frac{\eta}{8} \). Notice the query of \( \tilde{\alpha} \) is with error and success probability. We only need
\[
\epsilon_1 \sum_{s=1}^r |u_s| \frac{36}{\epsilon_1^2} \|6 \log_2 \left( \frac{16}{\eta} \right)\| \leq \frac{\epsilon}{2} \|\alpha\| \|x\|,
\]
\[
r \eta_1 \frac{36}{\epsilon_1^2} \|6 \log_2 \left( \frac{16}{\eta} \right)\| \leq \frac{\eta}{8}
\]
to fulfill the overall computing task. Notice \( \sum_{s=1}^r |u_s| \leq \sqrt{r} \|u\| \) and \( \alpha = R^T u \) We set
\[
\epsilon_1 = \frac{\epsilon \|x\|}{2 \sqrt{r} \frac{36}{\epsilon_1} \|6 \log_2 \left( \frac{16}{\eta} \right)\|}.
\]
\[ \eta_1 = \frac{\eta}{8r \left( \frac{\eta}{\epsilon} \right) \cdot \left[ 6 \log_2 \left( \frac{d}{\eta} \right) \right] } \]

And the overall time complexity for computing \( x^T X \alpha \) is

\[
T_i = O \left( \frac{1}{\epsilon^2} \log \frac{1}{\eta} \left( \log_2 (mn) + r k Q(R) \right) \right)
= O \left( \frac{1}{\epsilon^2} \log \frac{1}{\eta} \left( \log_2 (mn) + r k \log_2 \left( \frac{2}{\eta_1} \right) \log_2 (mn) \right) \right).
\]

VI. EXPERIMENTS

In this section, we demonstrate the proposed quantum-inspired SVM algorithm in practice by testing the algorithm on artificial datasets. The feasibility and efficiency of some other quantum-inspired algorithms (quantum-inspired algorithms for recommendation systems and linear systems of equations) on large datasets has been benchmarked, and the results indicate that quantum-inspired algorithms can perform well in practice under its specific condition: low rank, low condition number, and very large dimension of the input matrix [24]. Here we show the feasibility of the quantum-inspired SVM. Firstly, we test the quantum-inspired SVM algorithm on low-rank and low-rank approximated datasets and compare it to an existing classical SVM implementation. Secondly, we discuss the characteristics of the algorithm by analyzing its dependence on the parameters and datasets. In our experiment, we use the arborescent data structure instead of arrays for storage and sampling [24], making the experiment conducted in a more realistic scenario compared to the previous work [24]. All algorithms are implemented in Julia [22]. The source code and data are available at https://github.com/heloinrm/qisvm.

A. Experiment I: Comparison with LIBSVM

In this experiment, we test quantum-inspired SVM algorithm on large datasets and compare its performance to the well-known classical SVM implementation LIBSVM [33].

We generate datasets of size 10000×11000, which represent 11000 vectors (6000 vectors for training and 5000 vectors for testing) with length 10000. All the data vectors in training and testing sets are chosen uniformly at random from the generated data matrix, so that they are statistically independent and identically distributed. We test quantum-inspired SVM and LIBSVM on two kinds of datasets: low-rank datasets (rank=1) and high-rank but low-rank approximated datasets (rank=10000). Each scenario is repeated for 5 times. The construction method for data matrices is described in Appendix [3]. And the parameters for quantum-inspired SVM are chosen as \( \epsilon = 5, \eta = 0.1 \) and \( b = 1 \) (We explain the parameters and their setting in Experiment II).

The average classification rates are shown in Table [11] from which we observe the advantage of quantum-inspired SVM on such low-rank approximated datasets (on average about 5% higher). We also find that both quantum-inspired SVM and LIBSVM performs better on low-rank datasets than low-rank approximated datasets.

B. Experiment II: Discussion on algorithm parameters

As analyzed in Section [IV] and Section [V] there are two main parameters for the quantum-inspired algorithm: relative error \( \epsilon \) and success probability \( 1-\eta \). Based on them we set subsampling size \( r, c \) and run the algorithm. However, for datasets that are not large enough, setting \( r, c \) by Equation (26) and Equation (27) is rather time costly. For instance, when the condition number of data matrix is 1.0, taking \( \eta = 0.1 \) and \( \epsilon = 5.0 \), theoretically, the \( r, c \) for 10000×10000 dataset should be set as 1656 and 259973 to assure that the algorithm calculates the classification expression with relative error less than \( \epsilon \) and success probability higher than \( 1-\eta \). For practical applications of not too large datasets, we set \( r, c \) as \( r = b[4 \log_2(2n/\eta)/\epsilon^2] \) and \( c = b[4 \log_2(2r/\eta)/\epsilon^2] \), in which \( b \) is the subsampling size control parameter. When \( b = 1 \), our practical choice of \( r, c \) assures the relative error of subsampling (Step 2 and Step 3 in Alg. [3]) won’t exceed \( \epsilon \) (guaranteed by Theorem 2).

In Experiment I, we took the practical setting of \( r, c \), where we already found advantage compared to LIBSVM. Our choice of \( \epsilon, \eta \) and \( b \) is \( \epsilon = 5, \eta = 0.1 \) and \( b = 1 \). Here, we test the algorithm on other choices of \( \epsilon, \eta \) and \( b \) and check the classification rate of the algorithm. We test each parameter choice for 50 times. The variation intervals of each parameter are \( \epsilon \) from 1 to 10, \( \eta \) from 0.1 to 1, and \( b \) from 1 to 10. The results are shown in Fig. [4]. We find the average classification rates of the algorithm in each experiment are close. We notice when using the practical \( r, c \), which are much smaller than the theoretical ones, the algorithm maintains its performance (classification rate around 0.90). This phenomenon indicates a gap between our theoretical analysis and the actual performance, as [24] reports “the performance of these algorithms is better than the theoretical complexity bounds would suggest”.

VII. DISCUSSION

In this section, we will present some discussions on the proposed algorithm. And we will also discuss the potential applications of our techniques to other types of SVMs, such as non-linear SVM and least square SVM, but more works on the complexity and errors are required in future work if we want to realize these extensions.

A. The cause of exponential speedup

An interesting fact is that we can achieve exponential speedup without using any quantum resources, such as superposition or entanglement. This is a somewhat confusing but reasonable result that can be understood as follows: Firstly, the advantage of quantum algorithms, such as HHL algorithm, is that high-dimensional vectors can be represented using only a few qubits. By replacing qRAM to the arborescent data structure for sampling, we can also represent the low-rank matrices by its normalized submatrix in a short time. By using the technique of sampling, large-size calculations are avoided, and we only need to deal with the problem that has the logarithmic size of the original data. Secondly, the relative error of matrix subsampling algorithm is minus-exponential on the matrix size, which ensures the effectiveness of such
TABLE II
THE AVERAGE VALUES AND STANDARD DEVIATIONS OF CLASSIFICATION RATES (%) OF qISVM AND LIBSVM IN FIVE EXPERIMENTS.

|                      | Testing Set          | Training Set          |
|----------------------|----------------------|-----------------------|
|                      | qISVM                | LIBSVM                | qISVM                | LIBSVM                |
| Low-rank             | 86.45 ± 3.17         | 91.46 ± 2.00          | 86.45 ± 3.64         | 86.45 ± 2.15          |
| Low-rank approximated| 89.82 ± 4.38         | 84.90 ± 3.20          | 89.92 ± 4.23         | 84.69 ± 2.87          |

Fig. 4. The average classification rate of quantum-inspired SVM algorithm with different parameters on the dataset with rank 1. Each point represents an average classification rate for 50 trials, and the error bar shows the standard deviation of the 50 trials. (a) Algorithm performance when the parameter $\epsilon$ is taken from 1 to 10. (b) Algorithm performance when the parameter $\eta$ is taken from 0.1 to 1. (c) Algorithm performance when the parameter $b$ is taken from 1 to 10.

logarithmic-complexity algorithm (e.g. Theorem 2 shows the error of matrix rows sampling).

B. Improving sampling for dot product

Remember in Alg. 1 we can estimate dot products for two vectors. However, it does not work well for all the conditions, like when $\|x\|$ and $\|y\|$ are dominated by one element. For randomness, Eq. (54) implies that we can apply a spherically random rotation $R$ to all $x$, which does not change the kernel matrix $K$, but will make all the elements in the dataset matrix be in a same distribution.

C. LS-SVM with non-linear kernels

In Section II we have considered the LS-SVM with the linear kernel $K = X^T X$. When data sets are not linear separable, non-linear kernels are usually needed. To deal with non-linear kernels with Alg. 3 we only have to show how to establish sampling access for the non-linear kernel matrix $K$ from the sampling access of $X$.

We first show how the sampling access of polynomial kernel $K_p(x_i, x_j) = (x_i^T x_j)^p$ can be established. The corresponding kernel matrix is $K_p = ((x_i^T x_j)^p)_{i=1,...,m, j=1,...,m}$.

We take

$$Z = (x_1^{op}, x_2^{op}, ..., x_m^{op}),$$

in which the $j$-column $Z_j$ is the $p$-th tensor power of $x_j$.

Notice that $Z^T Z = K_p$. Once we have sampling access of $Z$, we can sample $K_p$ as Step 2 and Step 3 in Alg. 3 do. The sampling access of $Z$ can be established by (The effectiveness of Alg. 4 is shown in Appendix C):

Algorithm 4 Polynomial kernel matrices sampling.

Input: The sampling access of $X$ in logarithmic time of $m$ and $n$.

Goal: Sample a column index $j$ from the column norm vector $(\|x_1\|, \|x_2\|, ..., \|x_m\|)$ of $Z$, and then sample a row index $i$ from column $x_j^{op}$ of $Z$.

1: Sample on column norm vector $(\|x_1\|, \|x_2\|, ..., \|x_m\|)$ of $X$ to get index $j$.
2: Query $\|x_j\|$ from $(\|x_1\|, \|x_2\|, ..., \|x_m\|)$. Calculate $\|x_j\|^p$.
3: Sample a real number $a$ uniformly distributed in $[0, 1]$. If $a \geq \|x_j\|^p$, go to Step 1. If not, output index $j$ as the column index and continue.

For general non-linear kernels, we note that they can always be approximated by linear combination of polynomial kernels (and thus can be sampled based on sampling access of polynomial kernels) the corresponding non-linear feature function is continuous. For instance, the popular radial basis function (RBF) kernel

$$K_{RBF}(x_i, x_j) = \exp(-\frac{\|x_i - x_j\|^2}{2\sigma^2})$$

can be approximated by

$$K_{RBF}(x_i, x_j) = \sum_{p=0}^{N} \frac{1}{p!} \left( -\frac{x_i^T x_j - 2x_j^T x_i + x_i^T x_i}{2\sigma^2} \right)^p$$
\[ \begin{align*}
&= \sum_{p=0}^{N} \left( \frac{-1}{2 \sigma^2} \right)^p \sum_{q,i=0}^{p} \binom{p}{q} K_q(x_i, x_i) \\
&+ (-2)^p \left( \frac{1}{p} \right) K_i(x_i, x_j) + \left( \frac{p}{p - q - l} \right) K_{p-q-l}(x_j, x_j). 
\end{align*} \]

D. General LS-SVM

In the former sections, we began with a LS-SVM with \( b = 0 \) and linear kernels in Section III. And we showed how the method can be extended to nonlinear kernels in Section VII-C. Finally, we deal with the last assumption \( b = 0 \). We show how a general LS-SVM can be tackled using techniques alike in Alg. 3:

A general LS-SVM equation \([26]\) is

\[ \begin{pmatrix}
0 \\
1 \\
K + \gamma^{-1}I
\end{pmatrix}
\begin{pmatrix}
b \\
a
\end{pmatrix}
= 
\begin{pmatrix}
0 \\
y
\end{pmatrix},
\]

in which \( K \) is the kernel matrix.

Equation \((8)\) can be solved as follows:

(i) Firstly, by methods in Section VII-C, we establish the sampling access for kernel matrix \( K \). Suppose a sampling outcome of \( K \) is \( K'' \).

(ii) Secondly, take

\[ A = \begin{pmatrix}
0 \\
1 \\
K + \gamma^{-1}I
\end{pmatrix}, \]

and

\[ A'' = \begin{pmatrix}
0 \\
1 \\
K'' + \gamma^{-1}I
\end{pmatrix}. \]

We establish the eigen relations between \( A \) and \( A'' \) by theorems which are similar to Theorem 2 and Theorem 4.

(iii) Once \( A \in \mathbb{R}^{m \times n} \) is subsampled to \( A'' \in \mathbb{R}^{r \times r} \), we can continue Step 3 of Alg. 3.

(iv) Once Equation \((8)\) is solved in Step 7 of Alg. 3 which means we can establish the query access for \( a \). According to Equation \(8\), \( b = y_j - x_j^T X a - \gamma^{-1} a_j \) for any \( j \) such that \( a_j \neq 0 \). We can then evaluate the classification expression \( y_j + (x_j - x_j^T X a) \gamma^{-1} a_j \) and make classification using Alg. 1.

There are two ways to find \( j \): One is executing the rejection sampling on \( a \) using Alg. 2. The other is checking if \( a_j = 0 \) after each sampling of \( X \) in Step 3 of Alg. 3.

VIII. Conclusion

We have proposed a quantum-inspired SVM algorithm that achieves exponential speedup over the previous classical algorithms. The feasibility of the proposed algorithm is demonstrated by experiments. Our algorithm works well on low-rank datasets or datasets that can be well approximated by low-rank matrices, which is similar with quantum SVM algorithm \([31]\) as "when a low-rank approximation is appropriate". Certain investigations on the application of such an algorithm are required to make quantum-inspired SVM operable in solving questions like face recognition \([25]\) and signal processing \([35]\).

We hope that the techniques developed in our work can lead to the emergence of more efficient classical algorithms, such as applying our method to support vector machines with more complex kernels \([26], [36]\) or other machine learning algorithms. The technique of indirect sampling can expand the application area of fast sampling techniques. And it will make contribution to the further competition between classical algorithms and quantum ones.

Some improvements on our work would be made in the future, such as reducing the conditions on the data matrix, further reducing the complexity, and tighten the error bounds in the theoretical analysis, which can be achieved through a deeper investigation on the algorithm and the error propagation process. The investigation of quantum-inspired non-linear SVMs and least squares SVM as discussed in Section VII also requires theoretical analysis and empirical evaluations.

We note that our work, as well as the previous quantum-inspired algorithms, are not intended to demonstrate that quantum computing is uncompetitive. We want to find out where the boundaries of classical and quantum computing are, and we expect new quantum algorithms to be developed to beat our algorithm.

APPENDIX A

PROOF OF THEOREMS IN IV

A. Proof of Theorem 2

Proof: We break the expression \( |V''^T A' V_j' - \delta_{ij} \sigma_j^2| \) into two parts,

\[ |V''^T A' V_j' - \delta_{ij} \sigma_j^2| \leq |V''^T (A' - A'') V_j'| + |V''^T A'' V_j' - \delta_{ij} \sigma_j^2|. \]

For the first item, because of the condition \( ||A' - A''|| \leq \beta \) and \( V_j' \) are normalized,

\[ |V''^T (A' - A'') V_j'| \leq ||V''^T|| \cdot ||(A' - A'') V_j'|| \leq \beta. \]

For the second item, because of the condition \( A'' = \sum_{l=1}^{k} \sigma_l^2 V''_l V''_l^T \),

\[ |V''^T A'' V_j' - \delta_{ij} \sigma_j^2| = 0. \]

In all,

\[ |V''^T A' V_j' - \delta_{ij} \sigma_j^2| \leq \beta. \]

The description above can be written in short as follows:

\[ |V''^T A' V_j' - \delta_{ij} \sigma_j^2| \leq |V''^T (A' - A'') V_j'| + |V''^T A'' V_j' - \delta_{ij} \sigma_j^2| \leq ||V''^T|| \cdot ||(A' - A'') V_j'|| \leq \beta. \]

B. Proof of Theorem 4

Proof: Denote \( \bar{V}_j = V_j - \delta_{ij} \) as \( \Delta_1 \), \( |V''_i A \bar{V}_j - \delta_{ij} \sigma_j^2| \) as \( \Delta_2 \). By definition, \( \bar{V}_i = \frac{1}{\sigma_i^2} R R^T V''_i \). Thus

\[ \Delta_1 = \frac{V''_i A R R^T V''_j - \delta_{ij} \sigma_j^4}{\sigma_i^2 \sigma_j^2}. \]

We break it into two parts:

\[ \Delta_1 \leq \frac{1}{\sigma_i^2 \sigma_j^2} \left( |V''_i A' V_j' - \delta_{ij} \sigma_j^4| + |V''_i (R R^T - A' A') V_j'| \right), \]
For the first item, we have
\[
|V''_i^T A'A'V''_j - \delta_{ij}\sigma_i^4| \\
=|V''_i^T (A' - A')^2 V''_j + V''_i^T (A' - A') A'V''_j| \\
+|V''_i^T A'(A' - A') V''_j| + |V''_i^T A'A' V''_j - \delta_{ij}\sigma_i^4| \\
\leq|V''_i^T (A' - A')^2 V''_j| + |V''_i^T (A' - A') A'V''_j| \\
+|V''_i^T A'(A' - A') V''_j| + |V''_i^T A'A' V''_j - \delta_{ij}\sigma_i^4| \\
\leq\beta^2 + \sigma_j^2\beta + \sigma_i^2\beta.
\]

The last step above used the same technique as the proof of Thm 3.

For the second item, we have
\[
|V''_i^T (R^T - A'A')V''_j| \leq \|R^T - A'A'\| \\
= \|X'^T XX^T X' - X'^T X'X'^T X'\| \\
\leq \|X'\|^2 \|XX^T - X'X'^T \|.
\]

Because \( \|X'\| \leq \|X'\|_F = \|X\|_F \),

we have
\[
|V''_i^T (R^T - A'A')V''_j| \leq \epsilon' \|X\|_F^2.
\]

In all, due to \( \sigma_i \geq \kappa \) \( \forall i \in \{1, \ldots, k\} \),

\[
\Delta_1 \leq \frac{1}{\sigma_i^2 \sigma_j^2} (\beta^2 + \sigma_j^2\beta + \sigma_i^2\beta + \epsilon' \|X\|_F^2) \\
\leq \kappa^2 \beta^2 + 2\kappa \beta + \kappa^2 \epsilon' \|X\|_F^2.
\]

By definition, \( \tilde{V}_i = \frac{1}{\sigma_i} R^T V''_i \). Thus
\[
\Delta_2 = \left| \frac{V''_i^T RAR^T V''_j - \delta_{ij}\sigma_i^6}{\sigma_i^2 \sigma_j^2} \right|.
\]

We break it into two parts:
\[
\Delta_2 \leq \frac{1}{\sigma_i^2 \sigma_j^2} (|V''_i^T (R^T A' - A') V''_j| \\
+ |V''_i^T A'A' V''_j - \delta_{ij}\sigma_i^6|)
\]

For the first item, we have
\[
|V''_i^T (R^T A' - A') V''_j| \\
\leq \|R^T - A'A'\| \\
\leq \|X'^T XX^T X'X'^T X'\| \\
\leq \|X\|_F^2 \|XX^T - X'X'^T \| + \|(XX^T - X'X'^T)X'X'^T \| \\
\leq 2\|X\|_F^2 \|X\|^2 \|XX^T - X'X'^T \| \\
\leq 2\|X\|_F^2 \epsilon'.
\]

For the second item, we have
\[
|V''_i^T A'A' V''_j - \delta_{ij}\sigma_i^6| \\
= |V''_i^T (A' - A') A'A' V''_j + V''_i^T A' A' V''_j| \\
+ |V''_i^T A' A' V''_j + V''_i^T A A' V''_j - \delta_{ij}\sigma_i^6| \\
\leq |V''_i^T (A' - A') A'A' V''_j| + |V''_i^T A'A' V''_j| \\
+ |V''_i^T A' A' V''_j + V''_i^T A A' V''_j - \delta_{ij}\sigma_i^6| \\
\leq \|A' - A'\| \|A'A'\| + \|A'A\| \|A' - A'\| + \|A'A\| \|A' - A'\| \\
\leq \|X\|_F^2 \|A' - A'\| + \|X\|_F^2 \|A' - A'\| \\
+ \|X\|_F^2 \|A' - A'\| \\
\leq \beta \|X\|_F^2.
\]

In all,
\[
\Delta_2 \leq \frac{1}{\sigma_i^2 \sigma_j^2} (2\|X\|_F^2 \epsilon' + \beta \|X\|_F^2) \\
\leq (2\epsilon' + \beta \|X\|_F^2) \|X\|_F^2 \kappa^2.
\]

C. Proof of Theorem 5

Proof: For \( V_i^T V_j - \delta_{ij} \) is elements of \( V^T V - I \) and \( V_i^T V_j - \delta_{ij} \leq \frac{1}{4k} \),

\[
\|V_i^T V - I\| \leq k \max \{|V_i^T V_j - \delta_{ij}|\} \leq \frac{1}{4}.
\]

Thus \( \|V_i^T V\| \) is invertible and

\[
\|V_i^T V\|^{-1} = 1/\|V_i^T V\| \leq 1/(1 - \|V_i^T V - I\|) = \frac{4}{3}.
\]

Take \( B = V \Sigma^{-2} V^T A - I_m, \) we have
\[
|V_i^T B V_j| = \left| \sum_{l=1}^k \frac{V_i^T \tilde{V}_l \cdot V_j^T A \tilde{V}_l}{\sigma_l^2} - V_i^T V_j \right|.
\]

We break it into two parts:
\[
|V_i^T B V_j| \leq \left| \sum_{l=1}^k \frac{V_i^T \tilde{V}_l \cdot (V_j^T A \tilde{V}_l - \delta_{lj}\sigma_l^2)}{\sigma_l^2} \right| + \left| \sum_{l=1}^k \tilde{V}_l \tilde{V}_l \delta_{lj} - V_i^T V_j \right|.
\]

The second item is zero because
\[
\left| \sum_{l=1}^k \tilde{V}_l \tilde{V}_l \delta_{lj} - V_i^T V_j \right| = |V_i^T V_j - V_i^T V_j|.
\]

The first item
\[
\left| \sum_{l=1}^k \frac{V_i^T \tilde{V}_l \cdot (V_j^T A \tilde{V}_l - \delta_{lj}\sigma_l^2)}{\sigma_l^2} \right| \leq \kappa \|V_j^T A \tilde{V}_l - \delta_{lj}\sigma_l^2\| \leq \kappa \|V_j^T A \tilde{V}_l\| \leq \kappa \left( \sum_{l=1}^k \|V_j^T \tilde{V}_l\| + \|V_j^T V_i\| \right) \leq \kappa \left( \sum_{l=1}^k \|V_j^T \tilde{V}_l\| + \|V_j^T V_i\| \right) \leq \frac{5}{4} \kappa \kappa.
\]

Thus \( \|V_i^T B V_j\| \leq \frac{5}{2} \kappa \kappa \) and \( \|V_i^T V_i\| \leq \frac{5}{2} \kappa \kappa. \) By Theorem 5,
\[
\|V \Sigma^{-2} V^T A - I_m\| = \|B\| \leq \|(V_i^T V_i)^{-1} \| \|V_i^T B V_j\| \leq \frac{5}{3} \kappa \kappa \xi.
\]
In our experiment, we constructed artificial datasets which are low-rank or can be low-rank approximated. Here we put up our construction method:

1. Firstly, we multiply a random matrix $A$ of size $n \times k$ with another random matrix $B$ of size $k \times m$. The elements in both of them are evenly distributed in $[-0.5, 0.5]$. Denote the multiplication outcome as $X$. Then the rank of $X$ is at most $k$.

2. We add turbulence to the matrix $X$ by adding a random number evenly distributed in $[-0.1x, 0.1x]$ to all the elements in $X$, in which $x$ is the average of all the absolute values of $X$. After adding turbulence, $X$ is no more low-rank but still low-rank approximated.

3. We normalize $X$ such that $X$ has operator norm 1.

4. We divide the column vectors of $X$ into two classes by a random hyperplane $w^T x = 0$ that passes the origin (By random hyperplane we mean the elements in $w$ are uniformly sampled from $[0, 1]$ at random.), while making sure that both classes are not empty.

5. Since now we have $m$ linear-separable labeled vectors, each with length $n$. We choose uniformly at random $m_1$ of them for training, and let the other $m_2 = m - m_1$ for testing, while making sure that the training set includes vectors of both classes.

The goal of Alg. 4 is to sample a column index and a row index from $Z$. We show it achieves the goal.

Step 1-3 are for sampling out the column index. They are essentially Alg. 2 with $A = \text{Diag}(\|x_1\|^p, \ldots, \|x_m\|^p)$ and $b = (\|x_1\|, \ldots, \|x_m\|)$, which sample from the column norm vector $b = (\|x_1\|^p, \ldots, \|x_m\|^p)$ of $Z$ to get the column index $j$. We note that in practical applications, Step 1-3 can be adjusted for speedup, such as frugal rejection sampling suggested in [37].

Step 4 is for sampling out the row index. Suppose $l = \sum_{\tau=1}^P (i_{\tau} - 1) n^{p-\tau} + 1$. According the definition of tensor power, the $\tau$-th element of $x_j^{\otimes P}$ is

$$(x_j^{\otimes P})_{l} = \prod_{\tau=1}^P x_{i_{\tau}j}.$$ 

When Step 4 executes $p$ times of sampling on $x_j$, the probability of getting the outcome $i_1, i_2, \ldots, i_p$ is $\prod_{\tau=1}^P 1_{i_\tau j}$, which is exactly the probability of sampling out $(x_j^{\otimes P})_l$ in $x_j^{\otimes P}$. Thus we output index $l = \sum_{\tau=1}^P (i_{\tau} - 1) n^{p-\tau} + 1$.

ACKNOWLEDGMENT

The authors would like to thank Yi-Fei Lu for helpful discussions.
[19] S. Lloyd, M. Mohseni, and P. Rebentrost, “Quantum algorithms for supervised and unsupervised machine learning,” arXiv preprint, Nov. 2013. [Online]. Available: https://arxiv.org/abs/1307.0411

[20] E. Tang, “Quantum-inspired classical algorithms for principal component analysis and supervised clustering,” arXiv preprint, Oct. 2018. [Online]. Available: http://arxiv.org/abs/1811.00414

[21] A. Gilyén, S. Lloyd, and E. Tang, “Quantum-inspired low-rank stochastic regression with logarithmic dependence on the dimension,” arXiv preprint, Nov. 2018. [Online]. Available: http://arxiv.org/abs/1811.04909

[22] N.-H. Chia, H.-H. Lin, and C. Wang, “Quantum-inspired sublinear classical algorithms for solving low-rank linear systems,” arXiv preprint, 2018. [Online]. Available: https://arxiv.org/abs/1811.04859

[23] A. W. Harrow, A. Hassidim, and S. Lloyd, “Quantum algorithm for linear systems of equations,” Phys. Rev. Lett., vol. 103, no. 15, p. 150502, Oct. 2009.

[24] J. M. Arrazola, A. Delgado, B. R. Bardhan, and S. Lloyd, “Quantum-inspired algorithms in practice,” Quantum, vol. 4, p. 307, Aug. 2020. [Online]. Available: https://doi.org/10.22331/q-2020-08-13-307

[25] P. J. Phillips, “Support vector machines applied to face recognition,” in Advances Neural Inform. Processing Systems, vol. 48, no. 6241, Gaithersburg, MD, USA, Nov. 1999, pp. 803–809. [Online]. Available: https://doi.org/10.6028/nist.ir.6241

[26] J. A. K. Suykens and J. Vandewalle, “Least squares support vector machine classifiers,” Neural Process. Lett., vol. 9, no. 3, pp. 293–300, June 1999. [Online]. Available: https://doi.org/10.1023/A:1018628609742

[27] J. Platt, “Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines,” Apr. 1998. [Online]. Available: https://www.microsoft.com/en-us/research/publication/sequential-minimal-optimization-a-fast-algorithm-for-training-support-vector-machines/

[28] H. P. Graf, E. Cosatto, L. Bottou, I. Dourdanovic, and V. Vanpik, “Parallel Support Vector Machines: The Cascade SVM,” in Advances in Neural Information Processing Systems 17, L. K. Saul, Y. Weiss, and L. Bottou, Eds. MIT Press, 2005, pp. 521–528. [Online]. Available: http://papers.nips.cc/paper/2608-parallel-support-vector-machines-the-cascade-svm.pdf

[29] J. Xu, Y. Y. Tang, B. Zou, Z. Xu, L. Li, Y. Lu, and B. Zhang, “The generalization ability of svm classification based on markov sampling,” IEEE transactions on cybernetics, vol. 45, no. 6, pp. 1169–1179, 2014. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6881630

[30] B. Zou, C. Xu, Y. Lu, Y. Y. Tang, J. Xu, and X. You, “k-times markov sampling for svms,” IEEE transactions on neural networks and learning systems, vol. 29, no. 4, pp. 1328–1341, 2017. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7993056

[31] P. Rebentrost, M. Mohseni, and S. Lloyd, “Quantum support vector machine for big data classification,” Phys. Rev. Lett., vol. 113, p. 130503, Sept. 2014. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.113.130503

[32] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, “Julia: A fresh approach to numerical computing,” SIAM review, vol. 59, no. 1, pp. 65–98, 2017. [Online]. Available: https://doi.org/10.1137/141000671

[33] C.-C. Chang and C.-J. Lin, “LIBSVM: A library for support vector machines,” ACM Transactions on Intelligent Systems and Technology, vol. 2, pp. 27:1–27:27, 2011, software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm

[34] D. Achlioptas, F. McSherry, and B. Schölkopf, “Sampling techniques for kernel methods,” in Advances Neural Inform. Processing Systems, T. G. Dietterich, S. Becker, and Z. Ghahramani, Eds. Vancouver, British Columbia, Canada: MIT Press, Dec. 2002, pp. 335–342. [Online]. Available: https://papers.nips.cc/paper/2072-sampling-techniques-for-kernel-methods

[35] L. Wang, Support Vector Machines for Signal Processing, 1st ed. The Netherlands: Springer, Berlin, Heidelberg, 2005, ch. 15, pp. 321–342. [Online]. Available: https://doi.org/10.1007/978-1-095439

[36] L. Wang, Multiple Model Estimation for Nonlinear Classification, 1st ed. The Netherlands: Springer, Berlin, Heidelberg, 2005, ch. 2, pp. 49–76. [Online]. Available: https://doi.org/10.1007/978-1-095439

[37] I. L. Markov, A. Fatima, S. V. Isakov, and S. Boixo, “Quantum Supremacy Is Both Closer and Farther than It Appears,” arXiv preprint, Sep. 2018. [Online]. Available: http://arxiv.org/abs/1807.10749

Chen Ding received the B.S. degree from University of Science and Technology of China, Hefei, China, in 2019.

He is currently a graduate student in CAS Centre for Excellence and Synergetic Innovation Centre in Quantum Information and Quantum Physics. His current research interests include quantum machine learning, quantum-inspired algorithm designing and variational quantum computing.

Tian-Yi Bao received the B.S. degree from University of Science and Technology of China, Hefei, China, in 2018.

He is currently an Assistant Professor of Henan Key Laboratory of Quantum Information and Cryptography, Zhengzhou, China, and the Postdoctoral Fellow of University of Science and Technology of China, Hefei, China. He has authored or co-authored over 30 papers in refereed international journals and co-authored 1 book. His current research interests include secure cloud quantum computing, big data quantum computing, and the physical implementation of quantum computing architectures, in particular using linear optical and superconducting systems.