Quantum Support Vector Machine for Big Data Classification

Patrick Rebentrost,[1,*] Masoud Mohseni,[2] and Seth Lloyd[1,3,†]

[1]*Research Laboratory of Electronics, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, USA*

[2]*Google Research, Venice, California 90291, USA*

[3]*Department of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139, USA*

(Received 12 February 2014; published 25 September 2014)

Supervised machine learning is the classification of new data based on already classified training examples. In this work, we show that the support vector machine, an optimized binary classifier, can be implemented on a quantum computer, with complexity logarithmic in the size of the vectors and the number of training examples. In cases where classical sampling algorithms require polynomial time, an exponential speedup is obtained. At the core of this quantum big data algorithm is a nonsparse matrix exponentiation technique for efficiently performing a matrix inversion of the training data inner-product (kernel) matrix.

DOI: 10.1103/PhysRevLett.113.130503 PACS numbers: 03.67.Ac, 07.05.Mh

Machine learning algorithms can be categorized along a spectrum of supervised and unsupervised learning [1–4]. In strictly unsupervised learning, the task is to find structure such as clusters in unlabeled data. Supervised learning involves a training set of already classified data, from which inferences are made to classify new data. In both cases, recent "big data" applications exhibit a growing number of features and input data. A Support Vector Machine (SVM) is a supervised machine learning algorithm that classifies vectors in a feature space into one of two sets, given training data from the sets [5]. It operates by constructing the optimal hyperplane dividing the two sets, either in the original feature space or in a higher-dimensional kernel space. The SVM can be formulated as a quadratic programming problem [6], which can be solved in time proportional to $O(\log(\epsilon^{-1})\mathrm{poly}(N, M))$, with N the dimension of the feature space, M the number of training vectors, and ϵ the accuracy. In a quantum setting, binary classification was discussed in terms of Grover's search in [7] and using the adiabatic algorithm in [8–11]. Quantum learning was also discussed in [12,13].

In this Letter, we show that a quantum support vector machine can be implemented with $O(\log NM)$ run time in both training and classification stages. The performance in N arises due to a fast quantum evaluation of inner products, discussed in a general machine learning context by us in [14]. For the performance in M, we reexpress the SVM as an approximate least-squares problem [15] that allows for a quantum solution with the matrix inversion algorithm [16,17]. We employ a technique for the exponentiation of nonsparse matrices recently developed in [18]. This allows us to reveal efficiently in quantum form the largest eigenvalues and corresponding eigenvectors of the training data overlap (kernel) and covariance matrices. We thus efficiently perform a low-rank approximation of these matrices [Principal Component Analysis (PCA)]. PCA is a common task arising here and in other machine learning algorithms [19–21]. The error dependence in the training stage is $O(\mathrm{poly}(\epsilon_K^{-1}, \epsilon^{-1}))$, where ϵ_K is the smallest eigenvalue considered and ϵ is the accuracy. In cases when a low-rank approximation is appropriate, our quantum SVM operates on the full training set in logarithmic run time.

Support vector machine.—The task for the SVM is to classify a vector into one of two classes, given M training data points of the form $\{(\vec{x}_j, y_j) : \vec{x}_j \in \mathbb{R}^N, y_j = \pm 1\}_{j=1\ldots M}$, where $y_j = 1$ or -1, depending on the class to which $\vec{x}_j$ belongs. For the classification, the SVM finds a maximum-margin hyperplane with normal vector $\vec{w}$ that divides the two classes. The margin is given by two parallel hyperplanes that are separated by the maximum possible distance $2/|\vec{w}|$, with no data points inside the margin. Formally, these hyperplanes are constructed so that $\vec{w} \cdot \vec{x}_j + b \geq 1$ for $\vec{x}_j$ in the $+1$ class and that $\vec{w} \cdot \vec{x}_j + b \leq -1$ for $\vec{x}_j$ in the -1 class, where $b/|\vec{w}|$ is the offset of the hyperplane. Thus, in the primal formulation, finding the optimal hyperplane consists of minimizing $|\vec{w}|^2/2$ subject to the inequality constraints $y_j(\vec{w} \cdot \vec{x}_j + b) \geq 1$ for all j. The dual formulation [6] is maximizing over the Karush-Kuhn-Tucker multipliers $\vec{\alpha} = (\alpha_1, \ldots, \alpha_M)^T$ the function

$$L(\vec{\alpha}) = \sum_{j=1}^{M} y_j \alpha_j - \frac{1}{2} \sum_{j,k=1}^{M} \alpha_j K_{jk} \alpha_k, \tag{1}$$

subject to the constraints $\sum_{j=1}^{M} \alpha_j = 0$ and $y_j \alpha_j \geq 0$. The hyperplane parameters are recovered from $\vec{w} = \sum_{j=1}^{M} \alpha_j \vec{x}_j$ and $b = y_j - \vec{w} \cdot \vec{x}_j$ (for those j where $\alpha_j \neq 0$). Only a few of the α_j are nonzero: these are the ones corresponding to the $\vec{x}_j$ that lie on the two hyperplanes—the support vectors. We have introduced the kernel matrix, a central quantity in many machine learning problems [19,21], $K_{jk} = k(\vec{x}_j, \vec{x}_k) = \vec{x}_j \cdot \vec{x}_k$, defining the kernel function $k(x, x')$. More complicated nonlinear kernels and soft margins will be studied below. Solving the dual form involves evaluating the

$M(M-1)/2$ dot products $\vec{x}_j \cdot \vec{x}_k$ in the kernel matrix and then finding the optimal α_j values by quadratic programming, which takes $O(M^3)$ in the nonsparse case [22]. As each dot product takes time $O(N)$ to evaluate, the classical support vector algorithm takes time $O(\log(1/\epsilon) \times M^2(N+M))$ with accuracy ϵ. The result is a binary classifier for new data $\vec{x}$:

$$y(\vec{x}) = \mathrm{sgn}\left(\sum_{j=1}^{M} \alpha_j k(\vec{x}_j, \vec{x}) + b\right). \quad (2)$$

Quantum machine learning with the kernel matrix.—In the quantum setting, assume that oracles for the training data that return quantum vectors $|\vec{x}_j\rangle = 1/|\vec{x}_j| \sum_{k=1}^{N} (\vec{x}_j)_k |k\rangle$, the norms $|\vec{x}_j|$, and the labels y_j are given. The quantum machine learning performance is relative to these oracles and can be considered a lower bound for the true complexity [23]. One way of efficiently constructing these states is via quantum RAM, which uses $O(MN)$ hardware resources but only $O(\log MN)$ operations to access them; see [14,24]. Using the inner product evaluation of [14] to prepare the kernel matrix, we can achieve a run time for the SVM of $O(\log(1/\epsilon)M^3 + M^2 \log N/\epsilon)$. Classically, the inner product evaluation is $O(\epsilon^{-2}\mathrm{poly}(N))$ by sampling when the components of the $\vec{x}_j$ are distributed unevenly, for example, when a Fourier transform is part of the postprocessing step [23].

The kernel matrix plays a crucial role in the dual formulation Eq. (1) and the least-squares reformulation discussed in the next section. At this point we can discuss an efficient quantum method for direct preparation and exponentiation of the normalized kernel matrix $\hat{K} = K/\mathrm{tr}K$. For the preparation, first call the training data oracles with the state $1/\sqrt{M}\sum_{i=1}^{M}|i\rangle$. This prepares in quantum parallel the state $|\chi\rangle = 1/\sqrt{N_\chi}\sum_{i=1}^{M}|\vec{x}_i||i\rangle|\vec{x}_i\rangle$, with $N_\chi = \sum_{i=1}^{M}|\vec{x}_i|^2$, in $O(\log NM)$ run time. If we discard the training set register, we obtain the desired kernel matrix as a quantum density matrix. This can be seen from the partial trace $\mathrm{tr}_2\{|\chi\rangle\langle\chi|\} = 1/N_\chi \sum_{i,j=1}^{M}\langle\vec{x}_j|\vec{x}_i\rangle|\vec{x}_i||\vec{x}_j||i\rangle\langle j| = K/\mathrm{tr}K$. See [25] for an independent estimation of the trace of K.

For quantum mechanically computing a matrix inverse such as $\hat{K}^{-1}$, one needs to be able to enact $e^{-i\hat{K}\Delta t}$ efficiently. However, the kernel matrix $\hat{K}$ is not sparse for the application of the techniques in [27,28]. For the exponentiation of nonsparse symmetric or Hermitian matrices, a strategy was developed by us in [18]. We adapt it to the present problem. Adopting a density matrix description to extend the space of possible transformations gives, for some quantum state ρ, $e^{-i\hat{K}\Delta t}\rho e^{i\hat{K}\Delta t} = e^{-i\mathcal{L}_{\hat{K}}\Delta t}(\rho)$. The superoperator notation $\mathcal{L}_K(\rho) = [K, \rho]$ was defined. Applying the algorithm of [18] obtains

$$\begin{aligned} e^{-i\mathcal{L}_{\hat{K}}\Delta t}(\rho) &\approx \mathrm{tr}_1\{e^{-iS\Delta t}\hat{K} \otimes \rho e^{iS\Delta t}\} \\ &= \rho - i\Delta t[\hat{K}, \rho] + O(\Delta t^2). \end{aligned} \quad (3)$$

Here, $S = \sum_{m,n=1}^{M} |m\rangle\langle n| \otimes |n\rangle\langle m|$ is the swap matrix of dimension $M^2 \times M^2$. Equation (3) is the operation that is implemented on the quantum computer performing the machine learning. For the time slice Δt, it consists of the preparation of an environment state $\hat{K}$ (see above) and the application of the global swap operator to the combined system and environment state followed by discarding the environmental degrees of freedom. Equation (3) shows that enacting $e^{-i\hat{K}\Delta t}$ is possible with error $O(\Delta t^2)$. The efficient preparation and exponentiation of the training data kernel matrix, which appears in many machine learning problems [19,21], potentially enables a wide range of quantum machine learning algorithms. We now discuss a complete quantum big data algorithm.

Quantum least-squares support vector machine.—A key idea of this work is to employ the least-squares reformulation of the support vector machine developed in [15] that circumvents the quadratic programming and obtains the parameters from the solution of a linear equation system. The central simplification is to introduce slack variables e_j (soft margins) and replace the inequality constraints with equality constraints (using $y_j^2 = 1$):

$$y_j(\vec{w} \cdot \vec{x}_j + b) \geq 1 \rightarrow (\vec{w} \cdot \vec{x}_j + b) = y_j - y_j e_j. \quad (4)$$

In addition to the constraints, the implied Lagrangian function contains a penalty term $\gamma/2 \sum_{j=1}^{M} e_j^2$, where user-specified γ determines the relative weight of the training error and the SVM objective. Taking partial derivatives of the Lagrangian function and eliminating the variables $\vec{u}$ and e_j leads to a least-squares approximation of the problem:

$$F\begin{pmatrix} b \\ \vec{\alpha} \end{pmatrix} \equiv \begin{pmatrix} 0 & \vec{1}^T \\ \vec{1} & K + \gamma^{-1}\mathbb{1} \end{pmatrix}\begin{pmatrix} b \\ \vec{\alpha} \end{pmatrix} = \begin{pmatrix} 0 \\ \vec{y} \end{pmatrix}. \quad (5)$$

Here, $K_{ij} = \vec{x}_i^T \cdot \vec{x}_j$ is again the symmetric kernel matrix, $\vec{y} = (y_1, \ldots, y_M)^T$, and $\vec{1} = (1, \ldots, 1)^T$. The matrix F is $(M+1) \times (M+1)$ dimensional. The additional row and column with the $\vec{1}$ arise because of a nonzero offset b. The α_j take on the role as distances from the optimal margin and usually are not sparse. The SVM parameters are determined schematically by $(b, \vec{\alpha}^T)^T = F^{-1}(0, \vec{y}^T)^T$. As with the quadratic programming formulation, the classical complexity of the least-squares support vector machine is $O(M^3)$ [22].

For the quantum support vector machine, the task is to generate a quantum state $|b, \vec{\alpha}\rangle$ describing the hyperplane with the matrix inversion algorithm [16] and then classify a state $|\vec{x}\rangle$. We solve the normalized $\hat{F}|b, \vec{\alpha}\rangle = |\vec{y}\rangle$, where $\hat{F} = F/\mathrm{tr}F$ with $||F|| \leq 1$. The classifier will be determined by the success probability of a swap test between $|b, \vec{\alpha}\rangle$ and $|\vec{x}\rangle$. For application of the quantum matrix inversion algorithm, $\hat{F}$ needs to be exponentiated efficiently. The matrix $\hat{F}$ is schematically separated as $\hat{F} = (J + K + \gamma^{-1}\mathbb{1})/\mathrm{trF}$, with $J = \begin{pmatrix} 0 & \vec{1}^T \\ \vec{1} & 0 \end{pmatrix}$, and the Lie product formula allows for $e^{-i\hat{F}\Delta t} = e^{-i\Delta t \mathbb{1}/\mathrm{trF}} e^{-iJ\Delta t/\mathrm{trF}} e^{-iK\Delta t/\mathrm{trF}} + O(\Delta t^2)$. The matrix J is straightforward [27] (a "star" graph). The two nonzero eigenvalues of J are $\lambda_{\pm}^{\mathrm{star}} = \pm\sqrt{M}$ and the corresponding eigenstates are $|\lambda_{\pm}^{\mathrm{star}}\rangle = 1/\sqrt{2}(|0\rangle \pm (1/\sqrt{M})\sum_{k=1}^{M}|k\rangle)$. The matrix $\gamma^{-1}\mathbb{1}$ is trivial. For $K/\mathrm{tr}K$, proceed according to Eq. (3) by rescaling time by a factor $\mathrm{tr}K/\mathrm{tr}F = O(1)$ appropriately. This $e^{-i\hat{F}\Delta t}$ is employed conditionally in phase estimation.

The right-hand side $|\tilde{y}\rangle$ can be formally expanded into eigenstates $|u_j\rangle$ of $\hat{F}$ with corresponding eigenvalues λ_j, $|\tilde{y}\rangle = \sum_{j=1}^{M+1}\langle u_j|\tilde{y}\rangle|u_j\rangle$. With a register for storing an approximation of the eigenvalues (initialized to $|0\rangle$), phase estimation generates a state which is close to the ideal state storing the respective eigenvalue:

$$|\tilde{y}\rangle|0\rangle \to \sum_{j=1}^{M+1}\langle u_j|\tilde{y}\rangle|u_j\rangle|\lambda_j\rangle \to \sum_{j=1}^{M+1}\frac{\langle u_j|\tilde{y}\rangle}{\lambda_j}|u_j\rangle. \quad (6)$$

The second step inverts the eigenvalue and is obtained as in [16] by performing a controlled rotation and uncomputing the eigenvalue register. In the basis of training set labels, the expansion coefficients of the new state are the desired support vector machine parameters: $(C = b^2 + \sum_{k=1}^{M}\alpha_k^2)$,

$$|b, \vec{\alpha}\rangle = \frac{1}{\sqrt{C}}\left(b|0\rangle + \sum_{k=1}^{M}\alpha_k|k\rangle\right). \quad (7)$$

Classification.—We have now trained the quantum SVM and would like to classify a query state $|\vec{x}\rangle$. From the state $|b, \vec{\alpha}\rangle$ in Eq. (7), construct by calling the training data oracle

$$|\tilde{u}\rangle = \frac{1}{\sqrt{N_{\tilde{u}}}}\left(b|0\rangle|0\rangle + \sum_{k=1}^{M}\alpha_k|\vec{x}_k||k\rangle|\vec{x}_k\rangle\right), \quad (8)$$

with $N_{\tilde{u}} = b^2 + \sum_{k=1}^{M}\alpha_k^2|\vec{x}_k|^2$. In addition, construct the query state

$$|\tilde{x}\rangle = \frac{1}{\sqrt{N_{\tilde{x}}}}\left(|0\rangle|0\rangle + \sum_{k=1}^{M}|\vec{x}||k\rangle|\vec{x}\rangle\right), \quad (9)$$

with $N_{\tilde{x}} = M|\vec{x}|^2 + 1$. For the classification, we perform a swap test. Using an ancilla, construct the state $|\psi\rangle = 1/\sqrt{2}(|0\rangle|\tilde{u}\rangle + |1\rangle|\tilde{x}\rangle)$ and measure the ancilla in the state $|\phi\rangle = 1/\sqrt{2}(|0\rangle - |1\rangle)$. The measurement has the success probability $P = |\langle\psi|\phi\rangle|^2 = \frac{1}{2}(1 - \langle\tilde{u}|\tilde{x}\rangle)$. The inner product is given by $\langle\tilde{u}|\tilde{x}\rangle = 1/\sqrt{N_{\tilde{x}}N_{\tilde{u}}}(b + \sum_{k=1}^{M}\alpha_k|\vec{x}_k||\vec{x}|\langle\vec{x}_k|\vec{x}\rangle)$, which is $O(1)$ in the usual case when the α are not sparse. P can be obtained to accuracy ϵ by iterating $O(P(1-P)/\epsilon^2)$ times. If $P < 1/2$ we classify $|\vec{x}\rangle$ as $+1$; otherwise, -1.

Kernel matrix approximation and error analysis.—We now show that quantum matrix inversion essentially performs a kernel matrix principal component analysis and give a run time and error analysis of the quantum algorithm. The matrix under consideration, $\hat{F} = F/\mathrm{tr}F$, contains the kernel matrix $\hat{K}_\gamma = K_\gamma/\mathrm{tr}K_\gamma$ and an additional row and column due to the offset parameter b. In case the offset is negligible, the problem reduces to matrix inversion of the kernel matrix $\hat{K}_\gamma$ only. For any finite γ, $\hat{K}_\gamma$ is positive definite, and thus invertible. The positive eigenvalues of $\hat{F}$ are dominated by the eigenvalues of $\hat{K}_\gamma$. In addition, $\hat{F}$ has one additional negative eigenvalue which is involved in determining the offset parameter b. The maximum absolute eigenvalue of $\hat{F}$ is no greater than 1 and the minimum absolute eigenvalue is $\leq O(1/M)$. The minimum eigenvalue arises, e.g., from the possibility of having a training example that has (almost) zero overlap with the other training examples. Because of the normalization the eigenvalue will be $O(1/M)$ and, as a result, the condition number κ (the largest eigenvalue divided by the smallest eigenvalue) is $O(M)$ in this case. To resolve such eigenvalues would require exponential run time [16]. We define a constant ϵ_K such that only the eigenvalues in the interval $\epsilon_K \leq |\lambda_j| \leq 1$ are taken into account, essentially defining an effective condition number $\kappa_{\mathrm{eff}} = 1/\epsilon_K$. Then, the filtering procedure described in [16] is employed in the phase estimation using this κ_{eff}. An ancilla register is attached to the quantum state and appropriately defined filtering functions discard eigenvalues below ϵ_K when multiplying the inverse $1/\lambda_j$ for each eigenstate in Eq. (6). The desired outcome is obtained by postselecting the ancilla register.

The legitimacy of this eigenvalue filtering can be rationalized by a PCA argument. Define the $N \times M$ (standardized) data matrix $X = (\vec{x}_1, \ldots, \vec{x}_M)$. The $M \times M$ kernel matrix is given by $K = X^T X$. The $N \times N$ covariance matrix is given by $\Sigma = XX^T = \sum_{m=1}^{M}\vec{x}_m\vec{x}_m^T$. Often, data sets are effectively described by a few unknown factors (principal components) which admit a low-rank approximation for $\Sigma/\mathrm{tr}\Sigma$ with large $O(1)$ eigenvalues. The matrices K and Σ have the same nonzero eigenvalues. Keeping the large eigenvalues and corresponding eigenvectors of the kernel matrix by setting the cutoff $\epsilon_K = O(1)$ thus retains the principal components of the covariance matrix, i.e., the most important features of the data. See Supplemental Material for further discussion of the low-rank approximation [25]. $O(1)$ eigenvalues of $K/\mathrm{tr}K$ exist, for example, in the case of well-separated clusters with $O(M)$ vectors in them. A simple artificial example is $K = \mathbb{1}_{2\times 2} \otimes (\vec{1}\vec{1}^T)_{M/2\times M/2}$. The principal components of the kernel matrix are found in quantum parallel by phase estimation and the filtering procedure.

We continue with a discussion of the run time of the quantum algorithm. The interval Δt can be written as $\Delta t = t_0/T$, where T is the number of time steps in the phase estimation and t_0 is the total evolution time determining the error of the phase estimation [16]. The swap matrix used in Eq. (3) is 1-sparse and $e^{-iS\Delta t}$ is efficiently simulable in negligible time $\tilde{O}(\log(M)\Delta t)$ [28]. The $\tilde{O}$ notation suppresses more slowly growing factors, such as a $\log^* M$ factor [16,28]. For the phase estimation, the propagator $e^{-i\mathcal{L}_{\hat{F}}\Delta t}$ is enacted with error $O(\Delta t^2 ||\hat{F}||^2)$; see Eq. (3). With the spectral norm for a matrix A, $||A|| = \max_{|\vec{v}|=1}|A\vec{v}|$, we have $||\hat{F}|| = O(1)$. Taking powers of this propagator, $e^{-i\mathcal{L}_{\hat{F}}\tau\Delta t}$ for $\tau = 0, \ldots, T-1$ leads to an error of maximally $\epsilon = O(T\Delta t^2) = O(t_0^2/T)$. Thus, the run time is $T = O(t_0^2/\epsilon)$. Taking into account the preparation of the kernel matrix in $O(\log MN)$, the run time is thus $O(t_0^2\epsilon^{-1}\log MN)$. The relative error of λ^{-1} by phase estimation is given by $O(1/(t_0\lambda)) \leq O(1/(t_0\epsilon_K))$ for $\lambda \geq \epsilon_K$ [16]. If t_0 is taken $O(\kappa_{\text{eff}}/\epsilon) = O(1/(\epsilon_K\epsilon))$, this error is $O(\epsilon)$. The run time is thus $\tilde{O}(\epsilon_K^{-2}\epsilon^{-3}\log MN)$. Repeating the algorithm for $O(\kappa_{\text{eff}})$ times to achieve a constant success probability of the postselection step obtains a final run time of $O(\kappa_{\text{eff}}^3\epsilon^{-3}\log MN)$. To summarize, we find a quantum support vector machine that scales as $O(\log MN)$, which implies a quantum advantage in situations where classically $O(M)$ training examples and $O(N)$ samples for the inner product are required.

Nonlinear support vector machines.—One of the most powerful uses of support vector machines is to perform nonlinear classification [5]. Perform a nonlinear mapping $\vec{\phi}(\vec{x}_j)$ into a higher-dimensional vector space. Thus, the kernel function becomes a nonlinear function in $\vec{x}$:

$$k(\vec{x}_j, \vec{x}_k) = \vec{\phi}(\vec{x}_j) \cdot \vec{\phi}(\vec{x}_k). \quad (10)$$

For example, $k(\vec{x}_j, \vec{x}_k) = (\vec{x}_j \cdot \vec{x}_k)^d$. Now perform the SVM classification in the higher-dimensional space. The separating hyperplanes in the higher-dimensional space now correspond to separating nonlinear surfaces in the original space.

The ability of quantum computers to manipulate high-dimensional vectors affords a natural quantum algorithm for polynomial kernel machines. Simply map each vector $|\vec{x}_j\rangle$ into the d-times tensor product $|\phi(\vec{x}_j)\rangle \equiv |\vec{x}_j\rangle \otimes \cdots \otimes |\vec{x}_j\rangle$ and use the feature that $\langle\phi(\vec{x}_j)|\phi(\vec{x}_k)\rangle = \langle\vec{x}_j|\vec{x}_k\rangle^d$. Arbitrary polynomial kernels can be constructed using this trick. The optimization using a nonlinear, polynomial kernel in the original space now becomes a linear hyperplane optimization in the d-times tensor product space. Considering only the complexity in the vector space dimension, the nonlinear d-level polynomial quantum kernel algorithm to accuracy ϵ then runs in time $O(d\log N/\epsilon)$. Note that, in contrast to classical kernel machines, the exponential quantum advantage in evaluating inner products allows quantum kernel machines to perform the kernel evaluation directly in the higher-dimensional space.

Conclusion.—In this work, we have shown that an important classifier in machine learning, the support vector machine, can be implemented quantum mechanically with algorithmic complexity logarithmic in feature size and the number of training data, thus providing one example of a quantum big data algorithm. A least-squares formulation of the support vector machine allows the use of phase estimation and the quantum matrix inversion algorithm. The speed of the quantum algorithm is maximized when the training data kernel matrix is dominated by a relatively small number of principal components. We note that there exist several heuristic sampling algorithms for the SVM [29] and, more generally, for finding eigenvalues or vectors of low-rank matrices [30,31]. Information-theoretic arguments show that classically finding a low-rank matrix approximation is lower bounded by $\Omega(M)$ in the absence of prior knowledge [32], suggesting a similar lower bound for the least-squares SVM. Aside from the speedup, another timely benefit of quantum machine learning is data privacy [14]. The quantum algorithm never requires the explicit $O(MN)$ representation of all the features of each of the training examples, but it generates the necessary data structure, the kernel matrix of inner products, in quantum parallel. Once the kernel matrix is generated, the individual features of the training data are fully hidden from the user. Recently, quantum machine learning was discussed in [33,34]. In summary, the quantum support vector machine is an efficient implementation of an important machine learning algorithm. It also provides advantages in terms of data privacy and could be used as a component in a larger quantum neural network.

This work was supported by DARPA, NSF, ENI, AFOSR, the Google-NASA Quantum Artificial Intelligence Laboratory, and Jeffrey Epstein. The authors acknowledge helpful discussions with Scott Aaronson and Nan Ding.

[*]rebentr@mit.edu
[†]slloyd@mit.edu

[1] D. J. C. MacKay, *Information Theory, Inference and Learning Algorithms* (Cambridge University Press, Cambridge, England, 2003).
[2] E. Alpaydin, *Introduction to Machine Learning (Adaptive Computation and Machine Learning)* (MIT Press, Cambridge, MA, 2004).
[3] C. M. Bishop, *Pattern Recognition and Machine Learning* (Springer, New York, 2007).
[4] K. P. Murphy, *Machine Learning: A Probabilistic Perspective* (MIT Press, Cambridge, MA, 2012).
[5] C. Cortes and V. Vapnik, Mach. Learn. **20**, 273 (1995).
[6] S. Boyd and L. Vandenberghe, *Convex Optimization* (Cambridge University Press, Cambridge, England, 2004).
[7] D. Anguita, S. Ridella, F. Rivieccion, and R. Zunino, Neural Netw. **16**, 763 (2003).
[8] H. Neven, V. S. Denchev, G. Rose, and W. G. Macready, arXiv:0811.0416.

[9] H. Neven, V. S. Denchev, G. Rose, and W. G. Macready, arXiv:0912.0779.
[10] K. L. Pudenz and D. A. Lidar, Quantum Inf. Process. **12**, 2027 (2013).
[11] V. S. Denchev, N. Ding, S. V. N. Vishwanathan, and H. Neven, in *Proceedings of the 29th International Conference on Machine Learning, Edinburgh, Scotland, UK, 2012* (Omnipress, Wisconsin, 2012).
[12] M. Sasaki and A. Carlini, Phys. Rev. A **66**, 022303 (2002).
[13] R. A. Servedio and S. J. Gortler, SIAM J. Comput. **33**, 1067 (2004).
[14] S. Lloyd, M. Mohseni, and P. Rebentrost, arXiv:1307.0411.
[15] J. A. K. Suykens and J. Vandewalle, Neural Processing Letters **9**, 293 (1999).
[16] A. W. Harrow, A. Hassidim, and S. Lloyd, Phys. Rev. Lett. **103**, 150502 (2009).
[17] N. Wiebe, D. Braun, and S. Lloyd, Phys. Rev. Lett. **109**, 050505 (2012).
[18] S. Lloyd, M. Mohseni, and P. Rebentrost, Nat. Phys. **10**, 631 (2014)
[19] K.-R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf, IEEE Trans. Neural Networks **12**, 181 (2001).
[20] L. Hoegaerts, J. A. K. Suykens, J. Vandewalle, and B. D. Moor, in *Proceedings of IEEE International Joint Conference on Neural Networks, Budapest, Hungary, 2004* (IEEE, New York, 2004).
[21] T. Hofmann, B. Schölkopf, and A. J. Smola, Ann. Stat. **36**, 1171 (2008).
[22] The exponent 3 can be improved considerably; see D. Coppersmith and S. Winograd, J. Symb. Comput. **9**, 251 (1990).
[23] S. Aaronson, in *Proceedings of the 42nd ACM Symposium on Theory of Computing, Cambridge, MA, 2010* (ACM, New York, 2010).
[24] V. Giovannetti, S. Lloyd, and L. Maccone, Phys. Rev. Lett. **100**, 160501 (2008).
[25] See Supplemental Material http://link.aps.org/supplemental/10.1103/PhysRevLett.113.130503, which includes Ref. [26], for the estimation of the trace of the kernel matrix and details on the kernel matrix low-rank approximation.
[26] C. Ding and X. He, in *Proceedings of the 21st International Conference on Machine Learning, Banff, AB, Canada, 2004* (ACM, New York, 2004).
[27] A. Childs, Commun. Math. Phys. **294**, 581 (2010).
[28] D. W. Berry, G. Ahokas, R. Cleve, and B. C. Sanders, Commun. Math. Phys. **270**, 359 (2007).
[29] S. Shalev-Shwartz, Y. Singer, and N. Srebro, in *Proceedings of the 24th International Conference on Machine Learning, Corvallis, OR, 2007* (ACM, New York, 2007).
[30] E. Liberty, F. Woolfe, P.-G. Martinsson, V. Rokhlin, and M. Tygert, Proc. Natl. Acad. Sci. U.S.A. **104**, 20167 (2007).
[31] P. Drineas, R. Kannan, and M. W. Mahoney, SIAM J. Comput. **36**, 158 (2006).
[32] Z. Bar-Yossef, in *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, San Diego, 2003* (ACM, New York, 2003).
[33] N. Wiebe, A. Kapoor, and K. Svore, arXiv:1401.2142.
[34] G. D. Paparo, V. Dunjko, A. Makmal, M. A. Martin-Delgado, and H. J. Briegel, Phys. Rev. X **4**, 031002 (2014).