Classification of the MNIST data set with quantum slow feature analysis

Iordanis Kerenidis[*] and Alessandro Luongo[†]
CNRS, IRIF, Université Paris Diderot, 75013 Paris, France

(Received 16 November 2019; accepted 7 May 2020; published 22 June 2020)

Quantum machine learning is a research discipline intersecting quantum algorithms and machine learning. While a number of quantum algorithms with potential speedups have been proposed, it is quite difficult to provide evidence that quantum computers will be useful in solving real-world problems. Our work makes progress towards this goal. In this work we design quantum algorithms for dimensionality reduction and for classification and combine them to provide a quantum classifier that we test on the MNIST data set of handwritten digits. We simulate the quantum classifier, including errors in the quantum procedures, and show that it can provide classification accuracy of 98.5%. The running time of the quantum classifier is only polylogarithmic in the dimension and number of data points. Furthermore, we provide evidence that the other parameters on which the running time depends scale favorably, ascertaining the efficiency of our algorithm.

DOI: 10.1103/PhysRevA.101.062327

I. INTRODUCTION

Quantum computing has the potential to revolutionize information and communication technologies and one of the most promising areas is quantum machine learning. Recently, many quantum algorithms with potential speedups were proposed [1–7]. Nevertheless, whether quantum processing machines can be used in practice to solve efficiently and accurately real-world problems remains an open question. Answering this question consists in finding practical applications of quantum computing with a real economic and societal impact. Our work provides evidence that large-scale quantum computers with quantum access to data will indeed be useful in machine learning. We design two efficient quantum algorithms: one for dimensionality reduction and another for classification. In addition, we simulated their combined behavior on the commonly used MNIST data set of handwritten digits, showing that our classifier is able to provide accurate classification, comparable to classical algorithms.

Our first contribution consists of a quantum algorithm called quantum slow feature analysis (QSFA), a quantum method for dimensionality reduction. Dimensionality reduction is a technique used in machine learning in order to reduce the dimension of a data set while maintaining the most meaningful information contained therein. Dimensionality reduction algorithms are often used both to render the computational problem more feasible and to counterbalance the curse of dimensionality, i.e., the undesired property of some machine learning algorithms, whose performance deteriorates once the dimension in the feature space becomes too high. Intuitively, this is because if the data exist in an excessively-high-dimensional space, the informative power of the data points in the training set decreases, thus leading to a degradation in classification performance.

The second contribution is an algorithm for classification called the quantum Frobenius distance (QFD) classifier. It is a quantum method that classifies a new data point based on the average squared ℓ_2 distance between the test point and each labeled cluster. Being a very simple algorithm, we believe it can target quantum hardware in the early noisy intermediate-scale quantum architectures.

As the last contribution, the simulation results of our quantum classifier, combining QSFA and QFD, allows us to claim that the accuracy of the MNIST data set is around 98.5%, which is comparable to classical machine learning algorithms and better than many of the previous results registered in [8]. We also provide an estimate of the running time of our algorithm by calculating all the parameters that appear for the asymptotic running time and depend on the input data. Our estimate provides evidence that our quantum classifier can be more efficient than the corresponding classical one or more importantly that one can use our quantum classifier with higher input dimension, retaining efficiency and accuracy together.

The goal of this work is to assess the power of a quantum computer that can access classical data efficiently, in a similar manner to classical high-performance computing machines which by default possess a fast RAM. In other words, we assume the quantum computer can use an efficient quantum procedure to create quantum states corresponding to the classical data, for example, through a QRAM, a data structure that allows quantum access to classical data. Efficient algorithms for creating such QRAM circuits using quantum oracle access have been proposed [1,9,10]. Sometimes in the literature this data structure goes under the name of KP trees [11]. We can imagine the QRAM in two ways: as a quantum operator that allows classical data to be retrieved efficiently in superposition and as a particular format we impose on the classical data we store. In fact, the circuit for the QRAM of a data set X holds all the information needed to retrieve the matrix

[*]Also at QC Ware, 75013 Paris, France.
[†]Also at Atos Quantum Lab, 78340 Les Clayes-sous-Bois, France; aluongo@irif.fr

X and is efficiently built from X. Of course, we do not have such quantum computers or quantum RAM right now and there is a possibility that they may never be realized. Still, there are some proof-of-concept experiments in this direction [12–15]. We think of our results as motivation for actually building such quantum processing machines. We will provide more details about the QRAM model in the following sections. After a review of previous work and introduction of the techniques used, we provide a short description of the classical SFA technique and of the quantum procedures for performing linear algebra.

A. Previous work

We describe previous work on quantum classification. Quantum machine learning can be roughly divided in two different approaches. The first category of algorithms uses circuits of parametrized gates to perform machine learning tasks such as classification or regression [16–18]. In the training phase, the parameters of the circuit are learned using classical optimization techniques, where the function to optimize is a loss function calculated on the output of the quantum circuit [19]. Works in this direction are [20,21], with issues outlined in [22] and addressed in [23]. The second approach uses quantum computers to speed up the linear algebraic operations performed in classical machine learning, extending the famous Harrow-Hassidim-Lloyd algorithm [24]. The work of Rebentrost *et al.* [3] consists of an algorithm for a support vector machines classifier. In this class of algorithms, other quantum dimensionality reduction algorithms exist, principal component analysis [6] and linear and nonlinear Fisher discriminant analysis [25], which are based on Hamiltonian simulation techniques. Other possible approaches consist in using quantum annealing (see, e.g., [26–28]).

Recently, many results in classical machine learning have been obtained by dequantizing quantum machine learning algorithms [29,30], using techniques from randomized linear algebra and the Monte Carlo Markov chain [31]. Remarkably, this class of classical randomized algorithms works under assumptions similar to the ones in their quantum counterparts (i.e., they assume the ability to have query and weighted sample access to the data set). Similarly to what the QRAM does for quantum algorithms, these techniques store and precompute all the partial norms for the data set, which are later used in sampling procedures. As for the quantum case, the runtime of these classical algorithms is polylogarithmic in the dimensions of the data set. However, these algorithms are unfortunately impractical on interesting data sets: The most recent proposal for solving a linear system of equations has a runtime of $\tilde{O}(\kappa^{16}k^6\|A\|_F^6/\epsilon^6)$, where k is the rank, ϵ is the error in the solutions, κ is the condition number of the data set, and $\|A\|_F$ is the Frobenius norm of the data set. These algorithms have been implemented and benchmarked on real and synthetic data sets in [32]. Therein the authors conclude that these techniques cannot compare in terms of runtime to other approaches in classical machine learning, and therefore these algorithm are not likely to change the set of problems which is expected to be solved efficiently by quantum computers.

B. Classical slow feature analysis

Slow feature analysis was originally proposed as an online, nonlinear, and unsupervised algorithm [33,33,34]. It was motivated by the temporal slowness principle, a hypothesis for the functional organization of the visual cortex and possibly other sensory areas of the brain [35] and it was introduced as a way to model some transformation invariances in natural image sequences [36]. Slow feature analysis formalizes the slowness principle as a nonlinear optimization problem [37,38]. A prominent advantage of SFA compared to other algorithms is that it is almost hyperparameter-free. Another advantage is that it is guaranteed to find the optimal solution within the considered function space [39,40]. It has been shown that solving the optimization problem upon which SFA is based is equivalent to other dimensionality reduction algorithms, such as Laplacian eigenmaps [41] and the Fisher linear discriminant [42]; thus a quantum algorithm for SFA also provides algorithms for Laplacian eigenmaps and the Fisher linear discriminant. Our quantum algorithm runs in time polylogarithmically in the dimension and number of points in the data set, thus with a potential considerable speedup with respect to classical algorithms. With appropriate preprocessing, SFA can be used as a dimensionality reduction algorithm to improve the speed and classification accuracy in supervised machine learning [34,36,43,44]. The problem is formalized as follows. The input of the algorithm consists of vectors $x(i) \in \mathbb{R}^d$, $i \in [n]$. Each $x(i)$ belongs to one of K different classes. By definition, SFA algorithm computes the $K-1$ functions $g_j(x(i)) : \mathbb{R}^d \to \mathbb{R}$, $j \in [K-1]$, such that the output $y(i) = [g_1(x(i)), \ldots, g_{K-1}(x(i))]$ is very similar for the training samples that belong to the same class and largely different for samples of different classes. Once these functions are learned, they are used to map the training set in a low-dimensional vector space of dimension $K-1$. When a new data point arrives, it is mapped to the same vector space, where classification can be done with higher accuracy. We introduce the minimization problem as it is stated for classification [34]. For T_k the set of training elements of class k, let $a = \sum_{k=1}^{K} \binom{|T_k|}{2}$. For all $j \in [K-1]$ minimize

$$\Delta(y_j) = \frac{1}{a}\sum_{k=1}^{K} \sum_{\substack{s,t\in T_k \\ s<t}} [g_j(x(s)) - g_j(x(t))]^2, \qquad (1)$$

with the following constraints for all $v < j$: $\frac{1}{n}\sum_{k=1}^{K}\sum_{i\in T_k} g_j(x(i)) = 0$, $\frac{1}{n}\sum_{k=1}^{K}\sum_{i\in T_k} g_j(x(i))^2 = 1$, and $\frac{1}{n}\sum_{k=1}^{K}\sum_{i\in T_k} g_j(x(i))g_v(x(i)) = 0$. In order for the minimization problem to be feasible in practice, the g_j's are restricted to be linear functions w_j such that the output signal becomes $y(i) = [w_1^T x(i), \ldots, w_{K-1}^T x(i)]^T$ or else $Y = XW$, where $X \in \mathbb{R}^{n\times d}$ is the matrix with rows of the input samples and $W \in \mathbb{R}^{d\times(K-1)}$ the matrix that maps the input matrix X into a lower-dimensional output $Y \in \mathbb{R}^{n\times(K-1)}$. In case it is needed to capture nonlinear relations in the data set, one commonly performs a nonlinear polynomial expansion on the input data during the preprocessing. Usually, a polynomial expansion of degree 2 or 3 is sufficient, since polynomials of higher order might overfit. The constraint on the average and variance of the signal's component can be satisfied efficiently

by normalizing and scaling the input matrix X before solving the optimization problem. This assumption is simple to satisfy, as normalizing the input matrices of the data set is linear in the dimension of the problem. Taking all this into account, we can restate the definition of the Delta function as

$$\Delta(y_j) = \frac{w_j^T A w_j}{w_j^T B w_j}, \tag{2}$$

where the matrix B is called the covariance matrix of the data set and is defined as

$$B := \frac{1}{n}\sum_{i\in[n]} x(i)x(i)^T = X^T X \tag{3}$$

and the matrix A is called the derivative covariance matrix and is defined as

$$A := \frac{1}{a}\sum_{k=1}^{K}\sum_{\substack{i,i'\in T_k \\ i<i'}} [x(i)-x(i')][x(i)-x(i')]^T. \tag{4}$$

We rewrite A as $\frac{1}{a}\sum_{k=1}^{K}\dot{X}_k^T\dot{X}_k := \dot{X}^T\dot{X}$. Here $\dot{X}\in\mathbb{R}^{g\times d}$ is the matrix that has as row the difference between two vectors in X that belongs to the same class. We can approximate the matrix A by subsampling from all possible pairs $[x(i), x(i')]$ from each class. In our experiment we build $\dot{X}$ with a constant sample size of $g = 10^4$ derivatives from all the possible derivatives for a given class. It is not hard to see that the weight vectors w_j that correspond to the minima of Eq. (2) are the eigenvectors associated with the smallest eigenvalues of the generalized eigenvalue problem $AW = \Lambda BW$ [45,46], where $\Lambda = \mathrm{diag}[\lambda_1,\ldots,\lambda_n]$ is the diagonal matrix of eigenvalues and W is the matrix of generalized eigenvectors. Picking the $K-1$ smallest eigenvectors will allow us to create the $d\times(K-1)$ matrix W that will project our data to the slow feature space. In other words, SFA reduces to a generalized eigenvalue problem.

Computationally, the training part of the SFA algorithm has two steps. First, the matrix X is mapped to the matrix $Z = XB^{-1/2}$ of the whitened data. The whitening data matrix X consist in diagonalizing the covariance matrix X^TX of the data set and transforming the data such that the covariance matrix becomes the identity. Because of whitening, we reduce the generalized eigenvalue problem to a normal eigenvalue problem. Then Z is projected onto the space spanned by the eigenvectors associated with the $K-1$ smallest eigenvalues of the matrix $\dot{Z}^T\dot{Z}$. Here $\dot{Z}$ is defined similarly to $\dot{X}$, by sampling the pointwise differences of the whitened data. Because of this, we can redefine the derivative covariance matrix as $A := \dot{Z}^T\dot{Z} = (B^{-1/2})^T\dot{X}^T\dot{X}B^{-1/2}$.

We remark that the best classical algorithms for operating on matrices and performing all the necessary linear algebraic procedures are currently polynomial in the matrix dimensions. In fact, when the dimension of the input vectors becomes too large, procedures like full principal component analysis (PCA) or SFA become infeasible since their complexity is $\tilde{O}(\min(n^2d, d^2n))$ [47].

C. Quantum algorithms for linear algebra

In what follows we adopt the convention that the matrices stored in QRAM are prescaled, i.e., $\|M\|_2 = 1$, where $\|M\|_2$ is the spectral norm (see [10] for an efficient procedure for this normalization). In recent work [48,49], QRAM-based procedures for matrix multiplication and inversion were discovered with running time $\tilde{O}(\kappa(M)\mu(M)\log(1/\epsilon))$, where κ is the condition number of the matrix M (i.e., the ratio between the biggest and smallest singular value), ϵ is the error, and the parameter $\mu(M)$ is defined as

$$\mu(M) = \min_{p\in P}[\|M\|_F, \sqrt{s_{2p}(M)s_{2(1-p)}(M^T)}]$$

for $s_p(M) := \max_{i\in[n]}\|m_i\|_p^p$, where $\|m_i\|_p$ is the ℓ_p-norm of the ith row of M and P is a finite set of size $O(1)\in[0,1]$. Note that $\mu(M)\leqslant\|M\|_F\leqslant\sqrt{d}$, as we have assumed that $\|M\|_2\leqslant 1$. We see that it will be convenient to choose μ to be the maximum ℓ_1-norm of the rows of the matrix, i.e., $\mu(M) = \|M\|_\infty$. In these works, the notion of block encoding is used, which is shown to be equivalent to our notion of having an efficient data structure such as the one described in Appendix C. In order to get the polylogarithmic dependence on the precision parameter ϵ, Gilyén and co-workers showed how to perform the linear algebra procedures directly through the technique of qubitization and without explicitly estimating coherently the singular values. Another important advantage of the method is that it provides easy ways to manipulate sums or products of matrices. We start by stating more precisely the results in [48,49] about the quantum algorithms for linear algebraic procedures that we will use for proving the runtime and correctness of the QSFA algorithm.

Theorem 1 (matrix algebra [48,49]*).* Let $M := \sum_i \sigma_i u_i v_i^T \in \mathbb{R}^{d\times d}$ such that $\|M\|_2 = 1$ and a vector $x\in\mathbb{R}^d$ stored in QRAM. There exist quantum algorithms that with a probability of at least $1-1/\mathrm{poly}(d)$ return (i) a state $|z\rangle$ such that $||z\rangle - |Mx\rangle| \leqslant \epsilon$ in time $\tilde{O}(\kappa(M)\mu(M)\log(1/\epsilon))$ and (ii) a state $|z\rangle$ such that $||z\rangle - |M^{-1}x\rangle| \leqslant \epsilon$ in time $\tilde{O}(\kappa(M)\mu(M)\log(1/\epsilon))$. One can also get estimates of the norms with multiplicative error η by increasing the running time by a factor $1/\eta$.

Theorem 2 (matrix algebra on products of matrices [48,49]*).* Let $M_1, M_2\in\mathbb{R}^{d\times d}$ such that $\|M_1\|_2 = \|M_2\|_2 = 1$, $M = M_1M_2$, and a vector $x\in\mathbb{R}^d$ stored in QRAM. There exist quantum algorithms that with a probability of at least $1-1/\mathrm{poly}(d)$ return (i) a state $|z\rangle$ such that $||z\rangle - |Mx\rangle| \leqslant \epsilon$ in time $\tilde{O}(\kappa(M)[\mu(M_1)+\mu(M_2)]\log(1/\epsilon))$, (ii) a state $|z\rangle$ such that $||z\rangle - |M^{-1}x\rangle| \leqslant \epsilon$ in time $\tilde{O}(\kappa(M)[\mu(M_1)+\mu(M_2)]\log(1/\epsilon))$, and (iii) a state $|M^+_{\leqslant\theta,\delta}M_{\leqslant\theta,\delta}x\rangle$ in time $\tilde{O}(\frac{[\mu(M_1)+\mu(M_2)]\|x\|}{\delta\theta\|M^+_{\leqslant\theta,\delta}M_{\leqslant\theta,\delta}x\|})$. One can also get estimates of the norms with multiplicative error η by increasing the running time by a factor $1/\eta$.

II. QUANTUM SLOW FEATURE ANALYSIS

Our goal is to devise a quantum algorithm that maps a quantum state corresponding to the data X to a quantum state Y that represents the data in the slow feature space of the SFA algorithm. Formally, U_{QSFA} maps the state $|X\rangle := \frac{1}{\|X\|_F}\sum_{i=1}^n \|x(i)\||i\rangle|x(i)\rangle$ to $|Y\rangle := \frac{1}{\|Y\|_F}\sum_{i=0}^n \|y(i)\||i\rangle|y(i)\rangle$.

As the classical algorithm, QSFA is divided in two parts. In the first step we whiten the data, i.e., we map the state $|X\rangle$ to the state $|Z\rangle$, and in the second step we project $|Z\rangle$ onto the subspace spanned by the smallest eigenvectors of the whitened derivative covariance matrix $A = \dot{Z}^T\dot{Z}$. As we know classically, the data are whitened by multiplying X by the matrix $B^{-1/2} = (X^TX)^{-1/2}$; in other words, we build $|Z\rangle := |B^{-1/2}X\rangle$. The results in [48] state that the time to multiply by $B^{-1/2}$ is the same as the time to multiply by X and hence we can perform this multiplication using Theorem 1. Now we want to project this state onto the subspace spanned by the eigenvectors associated with the $K-1$ "slowest" eigenvectors (i.e., associated with the smallest singular values) of the whitened derivative covariance matrix $A := \dot{Z}^T\dot{Z}$, where $\dot{Z}$ is the whitened derivative matrix $\dot{Z} = \dot{X}B^{-1/2}$. The procedure follows the projection procedure that was used in [1] for recommendation systems, although the projection is now in the lowest eigenspectrum instead of the highest one. Let θ be a threshold value and δ a precision parameter. By $A_{\leqslant\theta,\delta}$ we denote a projection of the matrix A onto the vector subspace spanned by the union of the singular vectors associated with singular values that are smaller than θ and some subset of singular vectors whose corresponding singular values are in the interval $[\theta, (1+\delta)\theta]$. Again, we note that the eigenvalues of A are the squares of the singular values of $\dot{Z}$, and the two matrices share the same row space: $\dot{Z} = U\Sigma V^T$ and $A = V\Sigma^2V^T$. Note also that whitening the derivatives is equal to taking the derivatives of the whitened data. We can therefore use the quantum algorithms for linear algebra and state the following.

Theorem 3 (QSFA). Let $X = \sum_i \sigma_i u_i v_i^T \in \mathbb{R}^{n\times d}$ and its derivative matrix $\dot{X} \in \mathbb{R}^{n\log n\times d}$ stored in QRAM as described in Appendix C. Let $\epsilon, \theta, \delta, \eta > 0$. There exists a quantum algorithm that produces as output a state $|\overline{Y}\rangle$ with $\||\overline{Y}\rangle - |A^+_{\leqslant\theta,\delta}A_{\leqslant\theta,\delta}Z\rangle\| \leqslant \epsilon$ in time

$$\tilde{O}\left(\left(\kappa(X)\mu(X)\log(1/\varepsilon) + \frac{\mu(X)+\mu(\dot{X})}{\delta\theta}\right) \times \frac{\|Z\|}{\|A^+_{\leqslant\theta,\delta}A_{\leqslant\theta,\delta}Z\|}\right) \tag{5}$$

and an estimator $\overline{\|Y\|}$ with $|\overline{\|Y\|} - \|Y\|| \leqslant \eta\|Y\|$ with an additional $1/\eta$ factor in the running time.

A priori, one does not know the appropriate threshold value θ, but this can be found efficiently using a binary search. Note also that the output of the algorithm is not a classical description of the projected inputs but a quantum state that corresponds to these projected inputs, but this is sufficient for using the classification procedure described afterward. If necessary, we can recover classically the $K-1$ slow feature vectors using process tomography. We present the proof of Theorem 3 in Appendix A.

III. QUANTUM FROBENIUS DISTANCE CLASSIFIER

To perform the classification of the MNIST data set we propose the quantum Frobenius distance classifier, a quantum classification algorithm designed with the ease of implementation in mind. The QFD classifier assigns a test point $x(0)$ to the cluster k whose points have minimum normalized average squared ℓ_2 distance to $x(0)$. Let X_k be defined as the matrix whose rows are the vectors corresponding to the kth cluster and $|T_k|$ is the number of elements in that cluster. For the test point $x(0)$ and a class k, define the matrix $X(0) \in \mathbb{R}^{|T_k|\times d}$, which just repeats the row $x(0)$ $|T_k|$ times. Then we define

$$F_k(x(0)) = \frac{\|X_k - X(0)\|_F^2}{2\left[\|X_k\|_F^2 + \|X(0)\|_F^2\right]} \tag{6}$$

as the average normalized squared distance between $x(0)$ and the cluster k. Let $h : \mathcal{X} \to [K]$ be our classification function. Then the hypothesis of the class made by the QFD classifier on $x(0)$ is

$$h(x(0)) := \arg\min_{k\in[K]} F_k(x(0)). \tag{7}$$

Our algorithm, which is described in detail in Appendix D, estimates $F_k(x(0))$ efficiently, assuming the vectors and their norms are loaded into the quantum computer either from the QRAM or directly from some quantum process. For its implementation we use little more than the ability to query the QRAM and a Hadamard gate. In fact, we first create the following state by using one control qubit and the ability to efficiently create the quantum states corresponding to the data points:

$$\frac{1}{\sqrt{N_k}}\left(|0\rangle\sum_{i\in T_k}\|x(0)\||i\rangle|x(0)\rangle + |1\rangle\sum_{i\in T_k}\|x(i)\||i\rangle|x(i)\rangle\right). \tag{8}$$

Then we apply a Hadamard on the leftmost qubit and note that the probability of measuring 1 is exactly equal to $F_k(x(0))$. We can estimate the probability of 1 by measuring in the computational basis in time $\tilde{O}(\frac{1}{\eta^2})$, or in $\tilde{O}(\frac{1}{\eta})$ using amplitude amplification. We will see that in fact η does not have to be very small in order to classify correctly the MNIST data set after the dimensionality reduction (we can take $\eta = 1/10$), since the clusters are pretty well separated. The running time of this procedure is $\tilde{O}(\frac{K}{\eta})$, when we assume that the data are stored in QRAM or it can be efficiently created by a quantum procedure.

IV. QUANTUM CLASSIFIER

For each cluster k, we first use the QSFA procedure to project the input onto the slow feature space, where we use the QFD classifier. More precisely, for each class we use the QSFA procedure that maps $|X_k\rangle$ to a state $|\overline{Y_k}\rangle$, which is ε close to the state $|Y_k\rangle = \frac{1}{\|Y_k\|_F}\sum_{i\in T_k}\|y(i)\||i\rangle|y(i)\rangle$. We can also use the same procedure to construct the vector $|y(0)\rangle$ and hence construct a state equivalent to the state in Eq. (8) but in the slow feature space. Then the QFD algorithm assigns a class to $y(0)$ by finding the closest cluster with respect to the Frobenius distance in Eq. (6).

A. Analysis of the accuracy of the MNIST data set

Since currently available quantum hardware cannot run our algorithm or any significant part of it, we simulated QSFA and QFD on an Atos QLM with 6 TB of RAM. We simulated

our combined quantum classifier on the MNIST data set of handwritten digits, a standard data set used to benchmark machine learning algorithms. Details on the simulations can be found in Appendix B. Our software heavily relied on PYTHON packages for numerical computation and machine learning: SCIKIT-LEARN [50] and the SCIPY ecosystem [51]. To gauge the accuracy of our classification procedure, we first need to take into account the inherent errors in the quantum procedures (for example, in estimating the singular values) and second we need to test our classification algorithm, since our notion of distance is not among the ones used classically. Then we provide some estimation of the running time of the quantum algorithm, by computing the parameters that appear in the asymptotic complexity of the algorithm. Our estimation does not include terms that depend on the hardware implementations of the quantum algorithm, which are for the moment not clear, for example, the fact that a quantum step may be slower than a classical step or the possible overhead due to the error correcting codes. Our goal here is to see if, *a priori*, the quantum algorithm itself is more efficient than the classical one. If there is no speedup in our analysis, then one cannot hope to see a speedup in practice. If there is a significant speedup, then there is hope that part of this speedup (and which part depends on how good the whole technological stack will be, like hardware and error correcting codes) can be seen in practice.

The MNIST data set [8], first studied in [8,52], is composed of $n = 60\,000$ grayscale images of handwritten digits, 28×28 pixels. We will follow closely the methodology used to apply classical SFA to the MNIST data set, detailed in [34]. The methodology is as follows. First, the dimension of the data set is reduced with a PCA to something like 35 (or around 90, depending on the polynomial expansion degree we use in the following step). Second, a polynomial expansion of degree 2 or 3 is applied, hence making the dimension up to 10^4. For example, with polynomial expansion of degree 2 we mean that a vector $(x_1, x_2, \ldots, x_d)$ is mapped to $(x_1, \ldots, x_d, x_1^2, x_1x_2, \ldots, x_d^2)$. Third, the data are scaled to zero mean and unit variance. We tested our algorithm while increasing the dimension of the initial PCA on the training set, hoping the accuracy will get even better when we increase the dimension of the input.

In Fig. 1 we plot the accuracy (percentage of correctly classified digits from the test set) of the quantum classifier (QSFA plus QFD) using polynomial expansion of degrees 2 and 3, showing that we achieve very good accuracy. On the x axis we changed the initial resolution of the image using PCA. The highest performance accuracy was 98.5% with polynomial expansion of degree 3 and PCA dimension of 36, which took about an hour of simulation. Given the favorable dependence on the dimension, we expect a quantum computer to achieve even higher accuracy given a polynomial expansion of a higher degree and more data.

We also tested the accuracy of our classifier when instead of the condition number κ we use a condition threshold κ_t (i.e., we discard singular values of the matrices below a certain threshold, which we vary between 30 and 200). This usually improves the running time by a factor of 2 while keeping the accuracy unchanged. The threshold of the condition number of X is chosen to retain 99.5% of the singular values. In this

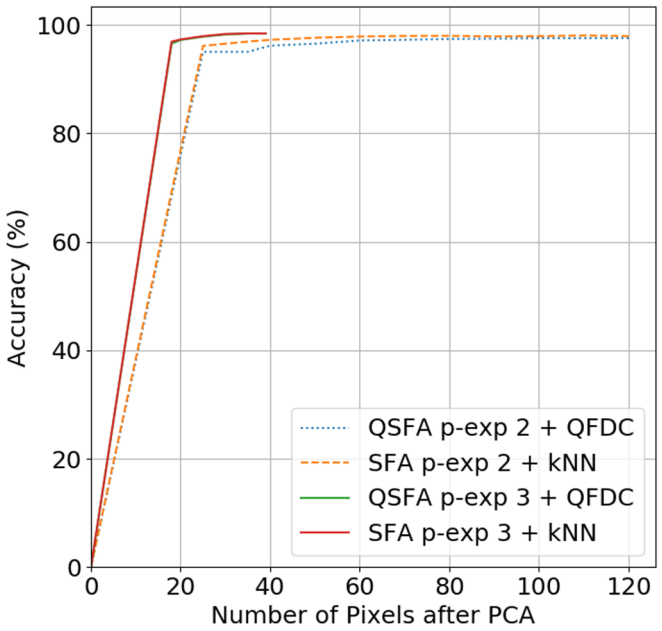

FIG. 1. Accuracy of our quantum classifier (QSFA with QFDC) versus the classical classifier (SFA with KNN) for polynomial expansions 2 and 3, using different numbers of pixels via PCA. For polynomial expansion 2, our quantum classifier (bottom dotted line) reaches the accuracy of the classical one (dashed line) for large enough dimensions. For polynomial expansion 3 (upper solid line), the two are almost indistinguishable. In this experiment, the error for the quantum classifier on the quantum linear algebra subroutines is $\epsilon = 10^{-5}$ and κ_t is less than 200.

case, the accuracy for polynomial expansion 2 remains practically unchanged, while for polynomial expansion 3 there is a small decline, with the maximum becoming 98.2% for PCA dimension 30. We can boost again the accuracy in the following way: Instead of running the classifier once, we run it a few times (around 10) and do a majority vote before labeling the new data point. This increases the accuracy of polynomial expansion 3 a bit; for example, for PCA 36, the accuracy goes from 97.7% to 97.9%. Note that while we tried to optimize the parameters in order to achieve the best possible accuracy, it is very probable there are even better parameters that in practice can be found by a hyperparameter tuning algorithm. This could further increase the accuracy of the quantum classifier.

B. Running time estimation

The asymptotic running time of the quantum classifier is given by the running time of the QSFA times a factor K/η^2 (or K/η with amplitude amplification) that comes from the QFD. For the MNIST data set, we estimated the value of the parameters that affects the runtime of the quantum classifier (training and testing part together). For instance, we measured the condition number of the matrices used in the quantum linear algebra operations. We also estimated the tolerable error in ϵ, δ, θ, and η such that the generalization error is comparable to that of classical algorithms. This was made possible by perturbing the model with some kind of noise, as described in Appendix B. Overall, putting the orders of all parameters together, the estimated asymptotic running time is

of the order of 10^7. Again, this is just an estimate of the steps of the quantum algorithm. More importantly, the behavior of the parameters as the dimension increases gives further evidence that the quantum classifier can be more efficient that a classical classifier, whose running time is of the order of 10^{13} for the same input dimension. Moreover, the fact that all parameters that appear in the running time of the quantum classifier seem to increase very slowly (or not at all) as we increase the number and dimension of the data points leads us to believe that one could still have an efficient quantum classifier with much a higher number and dimension of points, thus eventually providing much higher classification accuracy.

V. DISCUSSION

We have provided evidence that quantum computers with quantum access to data can be useful in solving real-world problems by designing an efficient classifier. We achieved accuracy comparable to the best classical algorithms, with improved running time. It would be interesting to see if the same performance is achievable also on different data sets; we do not see any reason why it would not be.

In one of the experiments our quantum classifier performs the training and testing part together, i.e., one composes the QSFA and QFD procedures, in contrast to the classical case where one first uses the training to obtain the model classically and then uses the model for the testing. On one hand, this allows us to quickly classify new data without having to wait for an extensive training period. In fact, according to our rough estimates, one can classify 10^6 images before the classical algorithm finishes its training part. On the other hand, if the testing part contains a very large number of data points (many orders of magnitude more than the training part), then one might want to use the quantum procedure to extract the classical SFA model, in other words, find the matrix W, such that $Y = XW$. For this, it suffices to use the QSFA algorithm with the initial state the totally mixed state, in which case the quantum output of the QSFA procedure is in fact the state that corresponds to the matrix W. Performing efficient tomography for this state, using the procedure in [53], we can accurately find the matrix W and store it in QRAM. Then the testing can be done classically in time $O(Kd)$.

We envision the utility of our classifier in conjunction with other quantum machine learning algorithms. For instance, projective simulation [54] is a reinforcement learning algorithm based on a random walk over a graph. The graph represents the memory of an agent that acts on a certain environment. The random walk starts from a node (or superposition of nodes) decided by an input-coupling function. Quantum SFA could be used to treat and preprocess high-dimensional input signals in agents that use projective simulation as the input-coupling function of external stimuli in the memory model of the agent. This would resemble even further what we currently believe to be the architecture of the brain, where SFA is used to model complex cells in the primary visual cortex (the first cortical area dedicated to visual processing [55]) and projective simulation is used to model high-level cognitive functions which emerge from a model of an episodic and compositional memory for the agent (so as to model creativity, curiosity, etc.) [56]. Note also that the quantum algorithm for SFA can also be used for classification via Fisher linear discriminant [42] and Laplacian eigenmaps [41].

Note also that it might be more important to see our quantum classifier not as a faster algorithm but as a way to increase the accuracy. As we said, the training stage is limited by the dimension of the input and the fact that our quantum classifier depends only polylogarithmically on the dimension and all other parameters remain stable when the dimension increases will enable us to use much higher dimension in the training stage, thus hopefully improving the accuracy of the classifier. Despite being an algorithm based on linear algebra, techniques like polynomial expansions can help in capturing nonlinearities hidden in the data, thus improving further the accuracy of the classifier. Alas, this comes at the cost of increasing the dimension of the data set and thus an increase of the runtime of classical algorithms. With quantum algorithms with polylogarithmic dependence on the dimension of the data set, we believe these techniques can be even more beneficial.

ACKNOWLEDGMENTS

We thank Anupam Prakash and András Gilyén for helpful discussions. This research was supported by QuantAlgo, Quantex, QuData, and ANR. The experiments were performed on an Atos QLM.

I.K. and A.L. contributed equally to the theoretical part of the work. A.L. made the simulations of the quantum algorithms.

The authors declare that there are no competing interests.

APPENDIX A: QUANTUM SLOW FEATURE ANALYSIS

In this Appendix we provide the details of the QSFA algorithm introduced in Theorem 3.

Quantum SFA consists of two steps. The first step is the whitening, which can be performed in time $\tilde{O}(\kappa(X)\mu(X)\log(1/\epsilon))$ and provide the state $|\overline{Z}\rangle$. It is simple to verify that creating a state $|Z\rangle$ of whitened data such that $Z^TZ = I$ can be done using quantum access just to the matrix X, as $Z = XB^{-1/2}$. The second step is the projection of whitened data in the slow feature space, which is spanned by the eigenvectors of $A = \dot{Z}^T\dot{Z}$. This matrix shares the same right eigenvectors of $\dot{X}B^{-1/2}$, which is simple to check that we can efficiently access using the QRAM constructions of X and $\dot{X}$. Using the algorithm for quantum linear algebra, we know that the projection (without the amplitude amplification) takes time equal to the ratio $\mu(X) + \mu(\dot{X})$ over the threshold parameter; in other words, it takes time $\tilde{O}(\frac{\mu(X)+\mu(\dot{X})}{\delta\theta})$. Finally, the amplitude amplification and estimation depend on the size of the projection of $|\overline{Z}\rangle$ onto the slow eigenspace of A; more precisely, it corresponds to the factor $\tilde{O}(\frac{\|\overline{Z}\|}{\|A^+_{\leqslant\theta,\kappa}A_{\leqslant\theta,\kappa}\overline{Z}\|})$. This term is roughly the same if we look at Z instead of $\overline{Z}$. Note also that Z is the whitened data, which means that each whitened vector should look roughly the same in each direction. This implies that the ratio should be proportional to the ratio of the dimension of the whitened data over the dimension of the output signal. Note that in the case of the MNIST data set this ratio is small enough.

Algorithm 1 Quantum slow feature analysis.

Require:

Matrices $X \in \mathbb{R}^{n\times d}$ and $\dot{X} \in \mathbb{R}^{n\times d}$ in QRAM, parameters $\epsilon, \theta, \delta, \eta > 0$.

Ensure:

A state $|\overline{Y}\rangle$ such that $||Y\rangle - |\overline{Y}\rangle| \leqslant \epsilon$, with $Y = A^{+}_{\leqslant\theta,\delta}A_{\leqslant\theta,\delta}Z$

1: Create the state

$$|X\rangle := \frac{1}{\|X\|_F}\sum_{i=1}^{n}\|x(i)\||i\rangle|x(i)\rangle\,.$$

2: (Whitening algorithm) Map $|X\rangle$ to $|\overline{Z}\rangle$ with $||\overline{Z}\rangle - |Z\rangle| \leqslant \epsilon$ and $Z = XB^{-1/2}$.

3: (Projection in slow feature space) Project $|\overline{Z}\rangle$ onto the slow eigenspace of A using threshold θ and precision δ, i.e., $A^{+}_{\leqslant\theta,\delta}A_{\leqslant\theta,\delta}\overline{Z}$.

4: Perform amplitude amplification and estimation on the register $|0\rangle$ with the unitary U implementing steps 1–3 to obtain $|\overline{Y}\rangle$ with $||\overline{Y}\rangle - |Y\rangle| \leqslant \epsilon$ and an estimate $\overline{\|Y\|}$ with multiplicative error η.

APPENDIX B: SIMULATION

The MNIST data set is a commonly used benchmark to test the validity of newly proposed classifiers. Classical classification techniques can achieve around 98%–99% accuracy, with neural network solutions exceeding 99% (the MNIST data set is quite simple and this is why it is often used as a first benchmark). As previously introduced in the main text, classical SFA has also been applied on the same data set with an accuracy of 98.5%, with initial PCA of dimension 35 and polynomial expansion of degree 3, and we will closely follow that classification procedure. Our goal is to study a quantum classifier with two properties: very good accuracy and efficiency. We detail our quantum classifier by going through the three parts in any classical classifier: preprocessing, training, and testing. The error in the whitening procedure has been simulated by adding noise from a truncated Gaussian distribution centered on each singular value with unit variance. For the error in the projection part, this comes only from potentially projecting on a different space than the one wanted. By taking the right θ (around 0.3 or 0.05 depending on the polynomial expansion) and a small enough δ (around $1/20$) for the error, we guarantee in practice that the projection is indeed on the smallest $K-1$ eigenvectors.

1. Data preprocessing

The MNIST data set is composed of $n = 60\,000$ images in the training set and 10 000 images in the test set, where each sample is a black and white image of a handwritten digit of 28×28 pixels. The methodology is as follows. First, the dimension of the images is reduced with a PCA to something like 35 (or around 90, depending on the polynomial expansion degree we use in the following step). Fortunately, efficient incremental algorithms for PCA exist, which do not require one to fully diagonalize a covariance matrix, and the running time depends on the number of dimensions required as output. Second, a polynomial expansion of degree 2 or 3 is applied, hence making the dimension up to 10^4. Third, the data are normalized so as to satisfy the SFA requirements of zero mean and unit variance. Overall, the preprocessing stage creates around $n = 10^5$ vectors $x(i)$, $i \in [n]$, of size roughly $d = 10^4$ and the running time of the preprocessing is $\tilde{O}(nd)$, with $nd \approx 10^9$. With a real quantum computer we would add a further step, which is to load the preprocessed data in the QRAM. This take only one pass over the data and creates a data structure (i.e., a circuit) which is linear in the size of the data. Hence, the overall preprocessing takes time of $\tilde{O}(nd)$.

2. Training

The classical SFA procedure outputs a small number $(K-1)$ of "slow" eigenvectors of the derivative covariance matrix, where K is the number of different classes and here $K = 10$. This is in fact the bottleneck for classical algorithms and this is why the dimension was kept below $d = 10^4$ with polynomial expansion, which still requires intensive high-performance computing calculations. Generically, the running time is between quadratic and cubic and hence of the order 10^{13}. Once these eigenvectors are found, each data point is projected onto this subspace to provide n vectors of $K-1$ dimensions which are stored in memory. As the points are labeled, we can find the centroid of each cluster. Note that at the end, the quantum procedure does not output a classical description of the eigenvectors, nor does it compute all vectors $y(i)$, as the classical counterpart could do. Nevertheless, given a quantum state $|x(i)\rangle$, it can produce the quantum state $|y(i)\rangle$ with high probability and accuracy.

3. Testing

For the testing stage, the classifier trained with the errors is used to classify the 10 000 images in the test set. Classically, the testing works as follows. One projects the test data point $x(0)$ onto the slow feature space to get a $(K-1)$-dimensional vector $y(0)$. Then a classification algorithm is performed, for example, the k-nearest neighbor (KNN) i.e., one finds the k-closest neighbors of $y(0)$ and assigns the label that appears the majority of times. The complexity of this step is $O(Kd)$ for the projection of the test vector, plus the time of the classification in the slow feature space. The KNN algorithm, for example, is linear in the number of data points times the dimension of the points $\tilde{O}(nd)$. In the nearest centroid algorithm (a supervised classification algorithm where the label of a new point is assigned to the cluster with the closest barycenter), if in the training stage we find the centroids, then classification can be done in time $\tilde{O}(Kd)$. In our final algorithm we perform

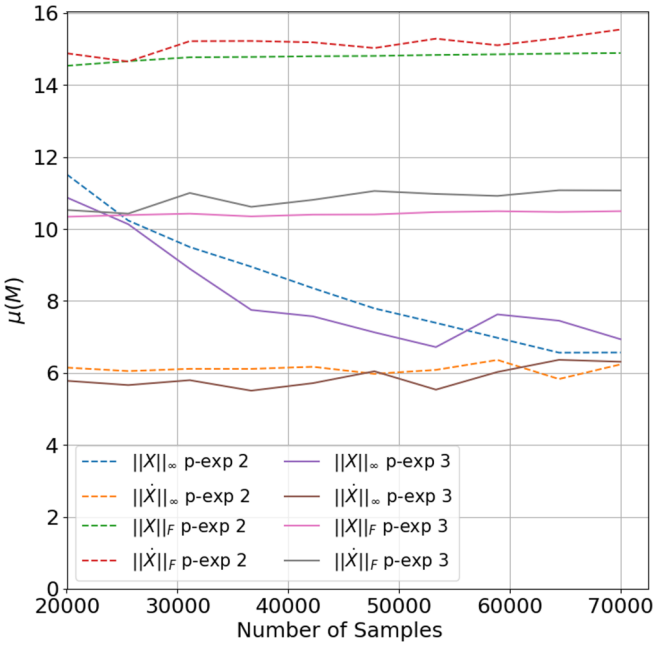

FIG. 2. Sensitivity analysis of the parameter μ of the data set while increasing n. The graph shows the value of the Frobenius norm and the ℓ_∞-norm as two options for μ. The two upper dotted lines represent the Frobenius norm of X and $\dot{X}$ (lower and upper line, respectively) for polynomial expansion of degree 2. The solid horizontal lines above 10 shows the trend of the Frobenius norm for the polynomial expansion of degree 3 of X and $\dot{X}$ (lower and upper line, respectively). The two lines at the bottom represent the ℓ_∞-norm for $\dot{X}$ for the polynomial expansion of degrees 2 (dashed) and 3 (solid), respectively. The central slanting lines are the ℓ_∞-norm of X, for polynomial expansion of degrees 2 (dashed) and 3 (solid). For the MNIST data set, we see that both the Frobenius norm and the ℓ_∞-norm are practically constant when we increase the number of data points in the training set.

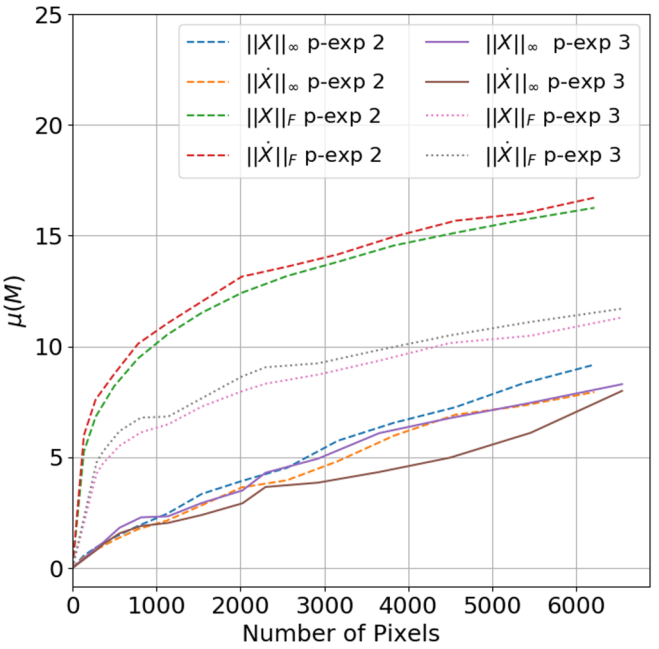

FIG. 3. Sensitivity analysis of the parameter μ for the matrices X and $\dot{X}$ while increasing d, the number of pixels for an image, by changing the PCA dimension, which is performed before the polynomial expansion. The plot shows the value of the Frobenius norm and the ℓ_∞-norm (i.e., max ℓ_1-norm of the rows) as two options for μ. The dashed lines are for the polynomial expansion of degree 2: The two upper lines are (from top to bottom) $\|\dot{X}\|_F$ and $\|X\|_F$ and the two lower dashed lines are (from top to bottom) $\|X\|_\infty$ and $\|\dot{X}\|_\infty$. The two dotted central lines track the Frobenius norm of the polynomial expansion of degree 3: $\|\dot{X}\|_F$ and $\|X\|_F$ (from top to bottom). The bottom solid lines represent the ℓ_∞-norm for the polynomial expansion of degree 3 of $\|\dot{X}\|$ and $\|X\|$ (from bottom to top). In general, the ℓ_∞-norm is always smaller than the Frobenius norm.

the training and testing together, i.e., using QSFA and QFD together.

4. Parameters of the experiment

We now estimate a number of parameters appearing in the running time of the quantum classifier.

a. Number and dimension of data points

For the MNIST we have that nd is of the order of 10^9 (including data points and derivative points).

b. The parameter μ for the matrices X and Ẋ

We analyze the parameters $\mu(X)$ and $\mu(\dot{X})$ as the number of data points in the training set and the dimension of the input vectors increase (PCA dimension plus polynomial expansion). We know that μ is bounded by the Frobenius norm of the matrix. We also look at the case where μ is defined as the maximum l_1-norm of the rows of the matrices, plotted in Fig. 2 (see also Fig. 3). Matrices are normalized to have spectral norm 1. The good choice of μ is practically constant as we increase the number of points in the data set. All the l_1-norms in the experiment are less than 11. We also plot the Frobenius norm and the maximum l_1-norm as the dimension of the vectors in the data set increases. While the Frobenius norm somewhat increases with the dimension, the maximum l_1-norm remains stable. This could be expected since in the preprocessing a PCA is done, making the input matrices in fact quite low rank. Indeed, after the polynomial expansion the Frobenius norm does not increase much since we only add higher-order terms; note that all entries of the matrices are smaller than 1, since $\|X\|_{\max} \leqslant \|X\|_2 \leqslant 1$. On the other hand, the scaling and normalization of X help keep the l_1-norm even lower. It is important to state here that one gains a factor 10^3 just by taking the correct quantum algorithm for performing linear algebra and not an off-the-shelf one. Such decisions will be crucial in reaching the real potential of quantum computing for machine learning applications.

c. Condition number for the matrix X

Figures 4 and 5 show us that condition number is rather stable, in fact decreasing. As explained, we do not need to have the real condition number in the running time but a threshold under which we ignore the smaller eigenvalues. In fact, retaining just 99.5% of the singular values does not

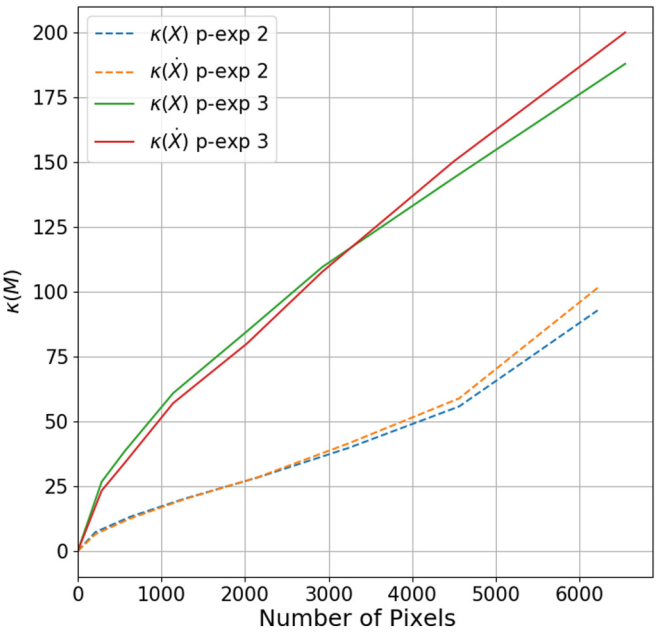

FIG. 4. Sensitivity analysis of the condition number of X and $\dot{X}$ while increasing d, the number of pixels, and discarding the 0 singular values. We plot the condition number of a polynomial expansion of degree 2: The upper dashed line is $\kappa(\dot{X})$ and the lower dashed line is $\kappa(X)$. The upper solid lines are for a polynomial expansion of degree 3. In both cases, the condition number of $\dot{X}$ dominates the condition number of X. For the case of a polynomial expansion of degree 3, this happens after approximately 3500 pixels.

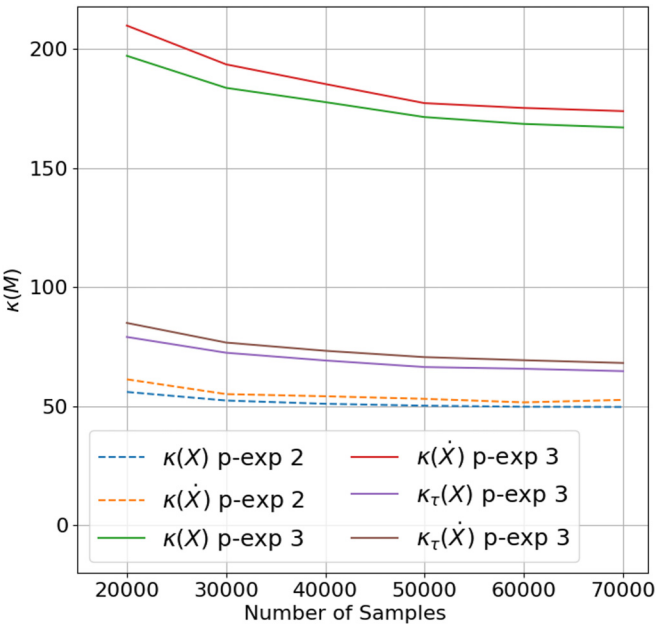

FIG. 5. Sensitivity analysis of the condition numbers (after discarding the 0 singular values) while increasing the size of the training set (by adding the test set). The first two lines from the top represent the polynomial expansion of degree 3, with $\kappa(\dot{X})$ bigger than $\kappa(X)$. The two central lines are the condition numbers for the polynomial expansion of degree 3, while discarding 0.5% of the smallest singular values [again $\kappa(\dot{X})$ is bigger than $\kappa(X)$]. The two bottom lines represent the condition numbers for polynomial expansion of degree 2. In all three cases, the condition number of $\dot{X}$ is bigger than the condition number of X.

considerably penalize the accuracy and achieves a behavior of growing much more slowly as we increase the dimension, with a value around 10^2.

d. Error parameters

There are four error parameters: ε for the matrix multiplication procedure, δ and θ for the projection procedure, and η for the estimate of the norms in the classification. For ε, it appears only within a logarithm in the running time, so we can take it to be rather small. For the projection, we take $\delta \approx 1/20$ and from the simulations we have that $\theta \approx 0.3$ for polynomial expansion of degree 2 and $\theta \approx 0.05$ for polynomial expansion of degree 3. Finally, it is enough to take $\eta = 1/10$. Note also that these parameters are pretty stable when increasing the dimension, as they only depend on whether we perform a polynomial expansion of 2 or 3.

e. Projection ratio

The ratio between the norm of the vectors in the whitened space over the projected vector in the slow feature space is well bounded. For an initial PCA dimension of 40 and polynomial expansion of degree 2, this ratio is 10 with variance 0.0022, while for a polynomial expansion of degree 3 and a PCA dimension of 30 it is 20 with 0.0007 variance.

Table I provides the exact values of all parameters in the experiment.

TABLE I. Accuracy of relevant experiments for various combination of classifiers and polynomial expansion. Here we have chosen $\epsilon/\kappa(X) = 10^{-7}$, $\delta = 0.054$, $\eta = 1/10$, and 10.000 derivatives per class.

d (PCA)	$\|X\|$	$\|\dot{X}\|$	$\|x(i)\|_1$	$\|\dot{x}(i)\|_1$	$\kappa(X)$	$\kappa(\dot{X})$	$\kappa_t(X)$	$\kappa_t(\dot{X})$	θ	$\%_t$	$\%$
					QSFA$_2$						
860 (40)	22	102	0.9	1.4	41	22	32	15	0.38	96.4	96.4
3220 (80)	41	197	2.7	3.8	119	61	65	30	0.32	97.3	97.4
4185 (90)	46	215	3.1	4.4	143	74	72	35	0.31	97.4	97.4
					QSFA$_3$						
5455 (30)	73	81	5.9	4.3	276	278	149	149	0.06	98.2	98.3
8435 (35)	96	102	7.8	5.7	369	389	146	156	0.05	97.5	98.4
9138 (36)	101	108	8.0	5.3	388	412	149	159	0.04	97.7	98.5

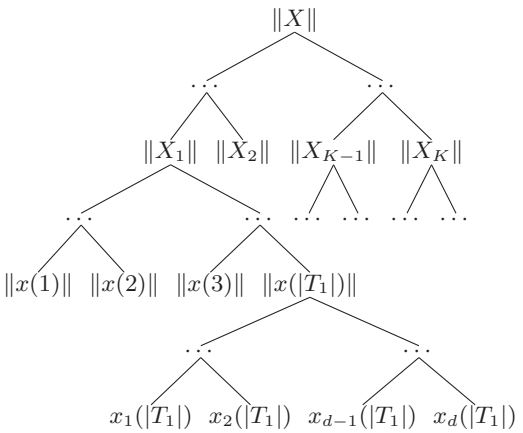

FIG. 6. The QRAM tree for matrices X_k and for the Frobenius norms of the submatrices of each class.

APPENDIX C: CONSTRUCTION OF THE QRAM

In this appendix we show how to construct the QRAM oracles needed in QSFA. A QRAM has been used in quantum algorithmics literature as a generic way of retrieving classical data and building a corresponding quantum state. The name QRAM is meant to evoke the way classical RAM addresses the data in memory using a tree structure. A quantum query is defined as $|i\rangle|0\rangle \to |i\rangle|b_i\rangle$ for $b_i \in \mathbb{R}$ and $i \in [N]$.

One can of course write down the real b_i with some precision δ using $\log 1/\delta$ bits. We extend this data structure to allow us to efficiently create superpositions corresponding to the rows of the matrices, states with amplitudes equal to the norms of the rows of the matrices, and also states corresponding to the inputs that belong to a specific class k. We show what our QRAM data structure looks like for the input matrix X (and similarly for the matrix $\dot{X}$) if the choice of μ is the Frobenius norm of the matrix. Each row of the matrix of the data set is encoded as a tree, where the leaves correspond to the matrix elements, while the intermediate nodes store the square amplitudes that correspond to their subtree.

We assume that the preprocessing (the optional PCA step, polynomial expansion, and removing the mean from the vectors and scaling the components to have unit variance) is performed classically, before storing the data in the QRAM. This takes $O(nd)$ time, which is upper bounded by $\tilde{O}(nd)$, the time needed to create this QRAM. Using the notation of the present paper, we have $X \in \mathbb{R}^{n\times d}$, $X = \cup_{i=1}^{K} X_i$, and $X_k \in \mathbb{R}^{|T_k|\times d}$. As we see in Fig. 6, all rows of the matrix X are saved as a tree with leaves the entries and the roots the corresponding norm $\|x(i)\|$. We arrange the rows per class and we join all trees corresponding to rows of a class k into a tree built on top of the individual trees, with leaves the norms of each row in the class (i.e., the roots of the previous trees) and the roots the norm of the class $\|X_k\|$. On top of these K trees, we build one last tree with the leaves the norms of each class (i.e., the roots of the previous trees) and the roots the norms of the entire matrix $\|X\|$. We also store the number of elements per class $T_k \in [K]$. This is said more concisely in the following corollary.

Corollary (QRAM for X_k). Let $X \in \mathbb{R}^{n\times d}$ and $X_k \in \mathbb{R}^{|T_k|\times d}$ for $k \in [K]$. There exists a data structure to store the rows of X such that (a) the size of the data structure is $O(nd\log^2(nd))$, (b) the time to store a row $x(i)$ is $O(d\log^2(nd))$ and the time to store the whole matrix X is thus $O(nd\log^2(nd))$, and (c) a quantum algorithm that can ask superposition queries of the data structure can perform in time polylog(nd) the unitaries (i) $U : |i\rangle|0\rangle \to |i\rangle|x(i)\rangle$ for $i \in [n]$, (ii) $V : |0\rangle \to \sum_{i\in[n]} \|x(i)\||i\rangle$, (iii) $U_k : |i\rangle|0\rangle \to |i\rangle|x(i)\rangle$ for $i \in T_k$ and all $k \in [K]$, and (iv) $V_k : |0\rangle \to \sum_{i\in T_k} \|x(i)\||i\rangle$ for all $k \in [K]$.

The procedure to create and store the matrix $\dot{X}$ is exactly the same as the procedure needed for X. Note that the classical matrix $\dot{X}_k$ is created in the following way. For each sample class T_k in the training set, we create $m = T_k\log(T_k)$ new derivative vectors $\dot{x}(i) := x(i) - x(j)$ with $i, j \in T_k$ by sampling from the uniform distribution m pairs of vectors with the same label.

APPENDIX D: QUANTUM FROBENIUS DISTANCE CLASSIFIER

We assume we can create a superposition of all vectors in the cluster as quantum states and have access to their norms, which can be achieved either using our QRAM or in the case in which the quantum states are efficiently constructible by quantum circuits. We define $N_k = \|X_k\|_F^2 + \|X(0)\|_F^2 = \|X_k\|_F^2 + |T_k|\|x(0)\|^2$. We give the steps to build an efficient procedure to estimate distances as Algorithm 2 and later describe how to use it to build a classifier. For the analysis

Algorithm 2 Quantum Frobenius distance estimator.

Require:

QRAM access to the matrix X_k of cluster k and to a test. vector $x(0)$. The error parameter $\eta > 0$.

Ensure:

$\overline{F_k(x(0))}$ such that $|F_k(x(0)) - \overline{F_k(x(0))}| < \eta$.

1: $s := 0$

2: **for** $r = O(1/\eta^2)$ **do**

3: Create the state

$$\tfrac{1}{\sqrt{N_k}}[\sqrt{|T_k|}\|x(0)\||0\rangle + \|X_k\|_F\,|1\rangle]\,|0\rangle|0\rangle$$

4: Apply the unitary that maps

$$|0\rangle|0\rangle \mapsto |0\rangle\,\tfrac{1}{\sqrt{|T_k|}}\sum_{i\in T_k}|i\rangle$$

and

$$|1\rangle|0\rangle \mapsto |1\rangle\,\tfrac{1}{\|X_k\|_F}\sum_{i\in T_k}\|x(i)\||i\rangle$$

to the first two registers to get

$$\tfrac{1}{\sqrt{N_k}}(\,|0\rangle\sum_{i\in T_k}\|x(0)\||i\rangle + |1\rangle\sum_{i\in T_k}\|x(i)\||i\rangle\,)\,|0\rangle\,.$$

5: Apply the unitary that maps

$$|0\rangle|i\rangle|0\rangle \mapsto |0\rangle|i\rangle|x(0)\rangle \quad\text{and}\quad |1\rangle|i\rangle|0\rangle \mapsto |1\rangle|i\rangle|x(i)\rangle\,.$$

6: Apply a Hadamard to the first register and measure it. If the outcome is $|1\rangle$ then $s := s + 1$.

7: **end for**

8: Output $\frac{s}{r}$.

of Algorithm 2, just note that the probability of measuring $|1\rangle$ in the final state is

$$\frac{1}{2N_k}\left(|T_k|\|x(0)\|^2 + \sum_{i\in T_k}\|x(i)\|^2 - 2\sum_{i\in T_k}\langle x(0), x(i)\rangle\right) = F_k(x(0)). \tag{D1}$$

By Hoeffding bounds, to estimate $F_k(x(0))$ with error η we would need $O(\frac{1}{\eta^2})$ samples. For the running time, we assume all unitaries are efficient either because the quantum states can be prepared directly by some quantum procedure or given that the classical vectors are stored in the QRAM. Hence the algorithm runs in time $\tilde{O}(\frac{1}{\eta^2})$. We can of course use amplitude estimation and save a factor of $1/\eta$. Depending on the application or the hardware, one may prefer to keep the quantum part of the classifier as simple as possible or optimize the running time by performing amplitude estimation. Given this estimator, we can now define the QFD classifier. For a test point, the classifier simply runs the distance estimation procedure for each cluster of vectors with the same label. Then the test point is assigned to the closest cluster. It is simple to see that the running time of this algorithm is $\tilde{O}(K/\eta^2)$ [or $\tilde{O}(K/\eta)$ using amplification techniques].

[1] I. Kerenidis and A. Prakash, Quantum recommendation systems, arXiv:1603.08675.

[2] S. Lloyd, S. Garnerone, and P. Zanardi, Quantum algorithms for topological and geometric analysis of data, Nat. Commun. **7**, 10138 (2016).

[3] P. Rebentrost, M. Mohseni, and S. Lloyd, Quantum Support Vector Machine for Big Data Classification, Phys. Rev. Lett. **113**, 130503 (2014).

[4] N. Liu and P. Rebentrost, Quantum machine learning for quantum anomaly detection, Phys. Rev. A **97**, 042315 (2018).

[5] N. Wiebe, D. Braun, and S. Lloyd, Quantum Algorithm for Data Fitting, Phys. Rev. Lett. **109**, 050505 (2012).

[6] S. Lloyd, M. Mohseni, and P. Rebentrost, Quantum principal component analysis, Nat. Phys. **10**, 631 (2013).

[7] A. M. Childs and N. Wiebe, Hamiltonian simulation using linear combinations of unitary operations, Quantum Inf. Comput. **12**, 901 (2012).

[8] Y. LeCun, C. Cortes, and C. J. C. Burges, The MNIST database of handwritten digits, 2010, http://yann.lecun.com/exdb/mnist/

[9] A. Prakash, Quantum algorithms for linear algebra and machine learning, Ph.D. thesis, University of California, Berkeley, 2014.

[10] I. Kerenidis and A. Prakash, Quantum gradient descent for linear systems and least squares, Phys. Rev. A **101**, 022316 (2020).

[11] P. Rebentrost and S. Lloyd, Quantum computational finance: Quantum algorithm for portfolio optimization, arXiv:1811.03975.

[12] N. Jiang, Y.-F. Pu, W. Chang, C. Li, S. Zhang, and L.-M. Duan, Experimental realization of 105-qubit random access quantum memory, npj Quantum Inf. **5**, 28 (2019).

[13] C. T. Hann, C.-L. Zou, Y. Zhang, Y. Chu, R. J. Schoelkopf, S. M. Girvin, and L. Jiang, Hardware-Efficient Quantum Random Access Memory with Hybrid Quantum Acoustic Systems, Phys. Rev. Lett. **123**, 250501 (2019).

[14] D. K. Park, F. Petruccione, and J.-K. K. Rhee, Circuit-based quantum random access memory for classical data, Sci. Rep. **9**, 3949 (2019).

[15] J. Bang, A. Dutta, S.-W. Lee, and J. Kim, Optimal usage of quantum random access memory in quantum machine learning, Phys. Rev. A **99**, 012326 (2019).

[16] G. Verdon, M. Broughton, and J. Biamonte, A quantum algorithm to train neural networks using low-depth circuits, arXiv:1712.05304.

[17] M. Schuld, A. Bocharov, K. Svore, and N. Wiebe, Circuit-centric quantum classifiers, Phys. Rev. A **101**, 032308 (2020).

[18] M. Benedetti, E. Lloyd, and S. Sack, Parameterized quantum circuits as machine learning models, Quantum Sci. Technol. **4**, 043001 (2019).

[19] M. Schuld, M. Fingerhuth, and F. Petruccione, Implementing a distance-based classifier with a quantum interference circuit, Europhys. Lett. **119**, 60002 (2017).

[20] J. S. Otterbach, R. Manenti, N. Alidoust, A. Bestwick, M. Block, B. Bloom, S. Caldwell, N. Didier, E. S. Fried, S. Hong, *et al.*, Unsupervised machine learning on a hybrid quantum computer, arXiv:1712.05771.

[21] E. Farhi and H. Neven, Classification with quantum neural networks on near term processors, arXiv:1802.06002.

[22] J. R. McClean, S. Boixo, V. N. Smelyanskiy, R. Babbush, and H. Neven, Barren plateaus in quantum neural network training landscapes, Nat. Commun. **9**, 4812 (2018).

[23] E. Grant, L. Wossnig, M. Ostaszewski, and M. Benedetti, An initialization strategy for addressing barren plateaus in parametrized quantum circuits, Quantum **3**, 214 (2019).

[24] A. W. Harrow, A. Hassidim, and S. Lloyd, Quantum Algorithm for Linear Systems of Equations, Phys. Rev. Lett. **103**, 150502 (2009).

[25] I. Cong and L. Duan, Quantum discriminant analysis for dimensionality reduction and classification, New J. Phys. **18**, 073011 (2016).

[26] M. Benedetti, J. Realpe-Gómez, R. Biswas, and A. Perdomo-Ortiz, Quantum-Assisted Learning of Hardware-Embedded Probabilistic Graphical Models, Phys. Rev. X **7**, 041052 (2017).

[27] C. Kaynak, Methods of combining multiple classifiers and their applications to handwritten digit recognition, Ph.D. thesis, Bogazici University, 1995.

[28] D. J. C. MacKay, *Information Theory, Inference, and Learning Algorithms* (Cambridge University Press, Cambridge, 2003).

[29] E. Tang, Quantum-inspired classical algorithms for principal component analysis and supervised clustering, arXiv:1811.00414.

[30] E. Tang, A quantum-inspired classical algorithm for recommendation systems, in *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, Association for Computing Machinery (STOC, New York, 2019), pp. 217–228.

[31] A. Frieze, R. Kannan, and S. Vempala, Fast monte-carlo algorithms for finding low-rank approximations, J. ACM **51**, 1025 (2004).

[32] J. M. Arrazola, A. Delgado, B. R. Bardhan, and S. Lloyd, Quantum-inspired algorithms in practice, arXiv:1905.10415.

[33] L. Wiskott, Learning invariance manifolds, Neurocomputing **26–27**, 925 (1999).

[34] P. Berkes, Pattern recognition with slow feature analysis, Cognitive Sciences EPrint Archive (CogPrints) 4104, 2005 (unpublished), available at http://cogprints.org/4104.

[35] L. Wiskott, P. Berkes, M. Franzius, H. Sprekeler, and N. Wilbert, Slow feature analysis, Scholarpedia **6**, 5282 (2011), revision 137965, available at http://www.scholarpedia.org/article/Slow_feature_analysis

[36] Z. Zhang and D. Tao, Slow feature analysis for human action recognition, IEEE Trans. Pattern Anal. Mach. Intell. **34**, 436 (2012).

[37] T. Blaschke and L. Wiskott, in *International Conference on Independent Component Analysis and Signal Separation*, edited by C. G. Puntonet and A. Prieto, Lecture Notes in Computer Science Vol. 3195 (Springer, Berlin, 2004), pp. 742–749.

[38] H. Sprekeler, T. Zito, and L. Wiskott, An extension of slow feature analysis for nonlinear blind source separation, J. Mach. Learn. Res. **15**, 921 (2014).

[39] A. N. Escalante-B and L. Wiskott, Slow feature analysis: Perspectives for technical applications of a versatile learning algorithm, KI-Künstliche Intelligenz **26**, 341 (2012).

[40] H. Sprekeler and L. Wiskott, Understanding slow feature analysis: A mathematical framework, Cognitive Sciences EPrint Archive (CogPrints) 6223, 2008 (unpublished), available at http://cogprints.org/6223.

[41] H. Sprekeler, On the relation of slow feature analysis and Laplacian eigenmaps, Neural Comput. **23**, 3287 (2011).

[42] S. Klampfl and W. Maass, in *Advances in Neural Information Processing Systems 22*, edited by Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta (MIT Press, Cambridge, 2009), pp. 988–996.

[43] X. Gu, C. Liu, and S. Wang, in *Biometric Recognition*, edited by Z. Sun, S. Shan, G. Yang, J. Zhou, Y. Wang, and Y. Yin (Springer, Berlin, 2013), p. 178.

[44] L. Sun, K. Jia, T.-H. Chan, Y. Fang, G. Wang, and S. Yan, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (IEEE, Piscataway, 2014), pp. 2625–2632.

[45] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, Springer Series in Statistics Vol. 1 (Springer, New York, 2009), pp. 337–387.

[46] A. H. Sameh and J. A. Wisniewski, A trace minimization algorithm for the generalized eigenvalue problem, SIAM J. Numer. Anal. **19**, 1243 (1982).

[47] D. A. Ross, J. Lim, R.-S. Lin, and M.-H. Yang, Incremental learning for robust visual tracking, Int. J. Comput. Vision **77**, 125 (2008).

[48] S. Chakraborty, A. Gilyén, and S. Jeffery, in *Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP)*, The power of block-Encoded matrix powers: Improved regression techniques via faster Hamiltonian simulation, (Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, 2019), pp. 33:1–33:14.

[49] A. Gilyén, Y. Su, G. H. Low, and N. Wiebe, Quantum singular value transformation and beyond: exponential improvements for quantum matrix arithmetics, in *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC)* (Association for Computing Machinery, New York, 2019), pp. 193–204.

[50] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, Scikit-learn: Machine Learning in Python, J. Mach. Learn. Res. **12**, 2825 (2011).

[51] E. Jones, T. Oliphant, P. Peterson *et al.*, SciPy: Open source scientific tools for Python (2001–2018) (accessed online 19 May 2018).

[52] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE **86**, 2278 (1998).

[53] I. Kerenidis and A. Prakash, A quantum interior point method for LPs and SDPs, arXiv:1808.09266.

[54] A. A. Melnikov, A. Makmal, V. Dunjko, and H. J. Briegel, Projective simulation with generalization, Sci. Rep. **7**, 14430 (2017).

[55] P. Berkes and L. Wiskott, Slow feature analysis yields a rich repertoire of complex cell properties, J. Vision **5**, 9 (2005).

[56] V. Dunjko and H. J. Briegel, Machine learning & artificial intelligence in the quantum domain: A review of recent progress, Rep. Prog. Phys. **81**, 074001 (2018).