ARTICLE OPEN

Nearest centroid classification on a trapped ion quantum computer

Sonika Johri[1]✉, Shantanu Debnath[1], Avinash Mocherla[2,3,4], Alexandros SINGK[2,3,5], Anupam Prakash[2,3], Jungsang Kim[1] and Iordanis Kerenidis[2,3,6]

Quantum machine learning has seen considerable theoretical and practical developments in recent years and has become a promising area for finding real world applications of quantum computers. In pursuit of this goal, here we combine state-of-the-art algorithms and quantum hardware to provide an experimental demonstration of a quantum machine learning application with provable guarantees for its performance and efficiency. In particular, we design a quantum Nearest Centroid classifier, using techniques for efficiently loading classical data into quantum states and performing distance estimations, and experimentally demonstrate it on a 11-qubit trapped-ion quantum machine, matching the accuracy of classical nearest centroid classifiers for the MNIST handwritten digits dataset and achieving up to 100% accuracy for 8-dimensional synthetic data.

npj Quantum Information (2021)7:122 ; https://doi.org/10.1038/s41534-021-00456-5

INTRODUCTION

Quantum technologies promise to revolutionize the future of information and communication, in the form of quantum computing devices able to communicate and process massive amounts of data both efficiently and securely using quantum resources. Tremendous progress is continuously being made both technologically and theoretically in the pursuit of this long-term vision.

On the one hand, quantum hardware is making considerable advances. Small quantum computers capable of running representative algorithms were first made available in research laboratories, utilizing both trapped ion[1,2] and superconducting qubits[3,4]. Performance comparisons among different quantum computer hardware have been made for running a host of quantum computing tasks[5,6]. A recent report on achieving quantum supremacy, where a task was performed by a quantum computer that cannot be simulated with any available classical computer[7], is an indication that powerful quantum computers will likely be available to researchers in the near future.

At the same time, considerable algorithmic work is underway in order to reduce the resources needed for implementing impactful quantum algorithms and bring them closer to the near-term intermediate scale quantum (NISQ) era. For example, more NISQ variants of amplitude estimation algorithms, a fundamental quantum procedure that is used in a large number of quantum algorithms, appeared recently[8–12].

In this work, we focus on quantum machine learning (QML). There are a number of reasons why machine learning is a good area to find applications of quantum computers. First, classical machine learning has proved to be an extremely powerful tool for a plethora of sectors, so enhancements from quantum computing will have a major impact. Second, we already know that fault-tolerant quantum computers with fast quantum access to classical data can provably offer advantages for many different applications[13–16]. Additionally, regularization techniques that artificially inject noise into the computation are often used to improve generalization performance for classical neural networks[17]. Thus, one might hope that noisy quantum computers are inherently better suited for machine learning computations than for other types of problems that need precise computations like factoring or search problems.

However, there are significant challenges to be overcome to make QML practical. First, most QML algorithms that offer considerable speedups assume that there are efficient ways to load classical data into quantum states. We address this bottleneck in this paper and describe ways to load a classical data point with logarithmic depth quantum circuits and using a number of qubits equal to the features of the data point. Another algorithmic bottleneck is that the solution output by the algorithm is often a quantum state from which one needs to extract some useful classical information. Here, we efficiently estimate the distance between two such states. Further, the efficiency of many of the quantum subroutines used in QML, in particular linear algebra subroutines depends on a large number of instance-specific parameters that need to be taken into account before claiming any speedups. In particular, most of these speedups will be polynomial and not exponential, which is also corroborated by quantum inspired algorithms[18]. We believe the way to describe these speedups is not to merely state them as exponential or polynomial, but to quantify the extent of speedup for each application and to see how these theoretical speedups translate in practice.

Another avenue for QML involves the use of parametrized quantum circuits as analogs of neural networks for supervised learning, in particular for classification[19,20] with the aim of increasing speed and accuracy. However, we neither have much theoretical evidence that such quantum architectures will be easily trained, nor can we perform large enough simulations to get any convincing practical evidence of their performance. Further, the time to train such quantum variational circuits can be quite large, both because of phenomena such as barren plateaus and also since designing the architectures, choosing cost-functions and initializing the parameters is far more complex and subtle than one may naively think[21–23].

[1]IonQ Inc, College Park, MD, USA. [2]QC Ware, Palo Alto, CA, USA. [3]QC Ware, Paris, France. [4]UCL Centre for Nanotechnology, London, UK. [5]Université Sorbonne Paris Nord, Villetaneuse, France. [6]CNRS University of Paris, Paris, France. ✉email: sonikajohri@gmail.com

Our work is a collaboration between quantum hardware and software teams that advances the state-of-the-art of QML implementations, bringing potential applications closer to reality. Even though the scale of the implementation remains a proof of concept, our work makes significant progress towards unblocking a number of theoretical and practical bottlenecks. In particular, we look at classification, one of the canonical problems in supervised learning with a vast number of applications. In classification, one uses a labeled dataset (for example, emails labeled as Spam or Not Spam) to fit a model which is then used to predict the labels for new data points. There are many different ways to perform classification that one can broadly place in two categories.

The first way is similarity-based learning, where a notion of similarity between data points is defined and points are classified together if they are similar. Well-known similarity-based algorithms are the Nearest Centroid, *k*-Nearest Neighbors, Support Vector Machines, etc. The second way is based on deep learning techniques. Here, the corpus of labeled data is used in order to train the weights of a neural network so that once trained it can infer the label of new data. Often, especially in cases where there is a large amount of data, neural networks can achieve better performance than more traditional similarity-based methods. On the other hand, similarity-based methods can offer other advantages, including provable performance guarantees and also properties like interpretability and transparency.

Here, we focus on demonstrating a quantum analog of the Nearest Centroid algorithm, a simple similarity-based classification technique, which is also used in clustering algorithms in unsupervised learning. The Nearest Centroid algorithm is a good baseline classifier that offers interpretable results, though its performance deteriorates when the data points are far away from belonging to convex classes with similar variances. The algorithm takes as input a number of labeled data points, where each data point belongs to a specific class. The model fitting part of the algorithm is very simple and it involves computing the centroids, i.e., the barycenters of each of the sets. Once the centroids of each class are found, then a new data point is classified by finding the centroid which is nearest to it in Euclidean distance and assigning the corresponding label. We demonstrate the quantum Nearest Centroid algorithm on up to 8 qubits of a trapped ion quantum processor and achieve accuracies comparable to corresponding classical classifiers on real datasets, as well as 100% accuracies on synthetic data. To our knowledge, this is the largest and most accurate classification demonstration on quantum computers.

We next describe some previous work on classification experiments on quantum computers, in particular with neural networks. There is a fast growing literature on variational methods for classification on small quantum computers[19,20,24–29] of which we briefly describe those that also include hardware implementations. In[28], the authors provide binary classification methods based on variational quantum circuits. The classical data is mapped into quantum states through a fixed unitary transformation and the classifier is a short variational quantum circuit that is learned through stochastic gradient descent. A number of results on synthetic data are presented showing the relation of the method to Support Vector Machines classification and promising performance for such small input sizes. In[29] the authors provide a number of different classification methods, based on encoding the classical data in separable qubits and performing different quantum circuits as classifiers, inspired by Tree Tensor Network and Multi-Scale Entanglement Renormalization Ansatz circuits. A 4-qubit hardware experiment for a binary classification task between two of the IRIS dataset classes was performed on an IBM machine with high accuracy. In[26], the authors performed 2-qubit experiments on IBM machines with the IRIS dataset. They reported high accuracy after training a quantum variational circuit for more than an hour using 5400 circuits and over 3 million total shots. A different approach using quantum annealers has also been used for generative learning[30].

RESULTS

Algorithm

We start by describing our data loaders[31]. Loading classical data as quantum states that can be efficiently used for further computation is an important step for QML applications, since most algorithms require efficient quantum access to data which is, and will likely remain, predominantly classical.

A data loader is a procedure that, given access to a classical data point $x = (x_1, x_2, \ldots, x_d) \in \mathbb{R}^d$, outputs a parametrized quantum circuit that prepares quantum states of the form

$$\frac{1}{\|x\|}\sum_{i=1}^{d} x_i|i\rangle. \tag{1}$$

Here, $|i\rangle$ is some representation of the numbers 1 through d. These states are called “amplitude encodings”. They are not the only possible way to represent classical data as quantum states but are by far the most interesting in terms of the quantum algorithms that can be applied to them. For example, they are the states one needs in order to start the quantum linear system solver procedure. To calculate the efficiency of our algorithms, we assume that quantum computers will have the ability to perform gates on different qubits in parallel. This is possible to achieve in most technologies for quantum hardware, including ion traps[32]. The Methods section discusses previous proposals for access to classical data.

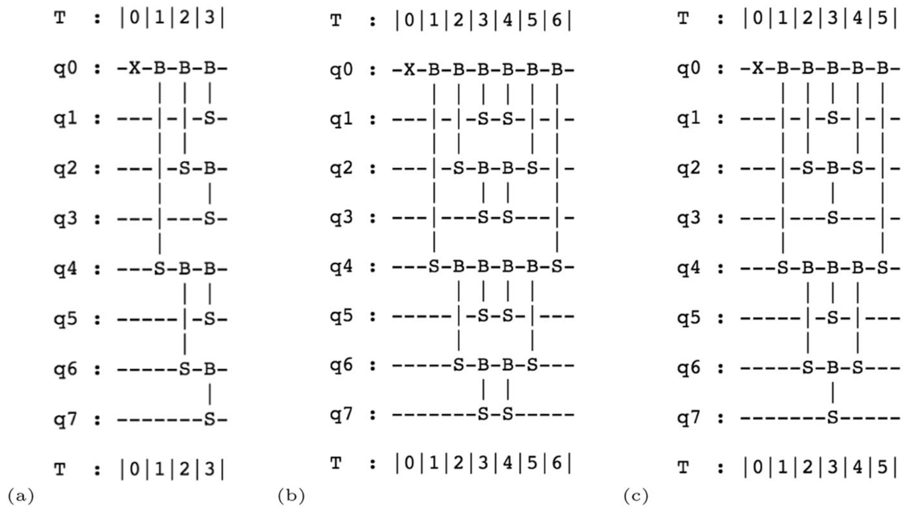
The loader we will use for our experiment is a “parallel” unary loader that loads a data point with d features, each of which can be a real number, with exactly d qubits, $d-1$ parametrized 2-qubit gates, and depth $\log d$. The parallel loader can be viewed as a part of a more extensive family of loaders with Q qubits and depth D, with $QD = O(d \log d)$. In particular, one can define an optimized loader with $2\sqrt{d}$ qubits and $\sqrt{d}\log d$ depth with $(d-1)$ two- and three-qubit gates in total[31] (see also Methods). Note that the number of qubits we use, one per feature, is the same as in most quantum variational circuit proposals (e.g., [19,29]).

We now describe the data loader construction. The first step is a procedure that given access to a classical data point $x = (x_1, x_2, \ldots, x_d) \in \mathbb{R}^d$, pre-processes the classical data efficiently, i.e., spending only $\tilde{O}(d)$ total time, in order to create a set of parameters $\theta = (\theta_1, \theta_2, \ldots, \theta_{d-1}) \in \mathbb{R}^{d-1}$, that will be the parameters of the $(d-1)$ two-qubit gates we will use in our quantum circuit. In the pre-processing, we also keep track of the norms of the vectors.

It is important here to notice that the classical memory is accessed once (we use read-once access to x) and the parameters θ of the quantum gates can be stored within the classical FPGA controller of the quantum computer, which means that if we need to perform many operations with the specific data point (which is the case here and, for example, in training neural networks), we do not need to access the memory of the classical computer again, we just need to re-run the quantum computer that already has the parameters of the quantum circuit in place in the controller.

The parameters, once computed, can be stored in the FPGA controlling the quantum computer. The classical control + the quantum computer are jointly referred to as the QPU or quantum processing unit. In that sense, the memory of the classical CPU does not need to be accessed again after the parameters have been calculated to load the data once.

The specific method for finding the parameters θ appears in Methods. Once we find the parameters that we need for our parametrized quantum circuit, we can define the architecture of



```
T  : |0|1|2|3|

q0 : -X-B-B-B-
        | | |
q1 : ---|-|-S-
        | |
q2 : ---|-S-B-
        |   |
q3 : ---|---S-
        |
q4 : ---S-B-B-
          | |
q5 : -----|-S-
          |
q6 : -----S-B-
            |
q7 : -------S-

T  : |0|1|2|3|
```

(a)

```
T  : |0|1|2|3|4|5|6|

q0 : -X-B-B-B-B-B-B-
        | | | | | |
q1 : ---|-|-S-S-|-|-
        | |     | |
q2 : ---|-S-B-B-S-|-
        |   | |   |
q3 : ---|---S-S---|-
        |         |
q4 : ---S-B-B-B-B-S-
          | | | |
q5 : -----|-S-S-|---
          |     |
q6 : -----S-B-B-S---
            | |
q7 : -------S-S-----

T  : |0|1|2|3|4|5|6|
```

(b)

```
T  : |0|1|2|3|4|5|

q0 : -X-B-B-B-B-B-
        | | | | |
q1 : ---|-|-S-|-|-
        | |   | |
q2 : ---|-S-B-S-|-
        |   |   |
q3 : ---|---S---|-
        |       |
q4 : ---S-B-B-B-S-
          | | |
q5 : -----|-S-|---
          |   |
q6 : -----S-B-S---
            |
q7 : -------S-----

T  : |0|1|2|3|4|5|
```

(c)

Fig. 1 Circuit constructions. a The data loader circuit for an 8-dimensional data point. The angles of the $RBS(\theta)$ gates starting from left to right and top to bottom correspond to $(\theta_1, \theta_2, \ldots, \theta_7)$. **b** The distance estimation circuit for two 8-dimensional data points. The circuit in time steps 0–3 corresponds to the parallel loader for the first data point and the circuit in time steps 4–6 corresponds to the inverse parallel loader circuit for the second data point (excluding the last X gate). The probability the first qubit is measured in state $|1\rangle$ is exactly the square of the inner product of the two data points. **c** The optimized distance estimation circuit for two 8-dimensional data points. Each pair of RBS gates that are applied to the same consecutive qubits in time steps 3 and 4 in the circuit of **b** are combined to one gate whose parameter θ is equal to $\theta_1 + \theta_2$, where θ_1 is the parameter of the first gate and θ_2 is the parameter of the second gate.

our quantum circuit. For convenience, we assume that d is a power of 2.

We will use a gate that has appeared with small variants with different names as partial SWAP, or fSIM, or Reconfigurable BeamSplitter, etc. We call this two-qubit parametrized gate $RBS(\theta)$ and we define it as

$$RBS(\theta) = \begin{pmatrix} ////1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2}$$

We can think of this gate as a simple rotation by an angle θ on the two-dimensional subspace spanned by the vectors $\{|10\rangle, |01\rangle\}$ and an identity in the other subspace spanned by $\{|00\rangle, |11\rangle\}$. We denote by $RBS^{\dagger}(\theta)$ the adjoint gate for which we have $RBS^{\dagger}(\theta) = RBS(-\theta)$.

We can now describe the circuit itself. We start by putting the first qubit in state $|1\rangle$, while the remaining $d-1$ qubits remain in state $|0\rangle$. Then, we use the first parameter θ_1 in an RBS gate in order to "split" this "1" between the first and the $d/2$-th qubit. Then, we use the next two parameters θ_2, θ_3 for the next layer of two RBS gates, where again in superposition we "split" the "1" into the four qubits with indices $(1, d/4, d/2, 3d/4)$ and this continues for exactly $\log d$ layers until at the end of the circuit we have created exactly the state

$$|x\rangle = \frac{1}{\|x\|}\sum_{i=1}^{d} x_i |e_i\rangle \tag{3}$$

where the states $|e_i\rangle$ are a unary representations of the numbers 1 to d, using d qubits. The circuit appears in Fig. 1a.

Let us make some remarks about the data loader circuits. First, the circuits use the optimal number of one-parameter gates, namely $d-1$, for loading any unit d-dimensional vector. Information-wise any circuit that can load all d-dimensional unit vectors exactly must have $d-1$ parameters. Second, we will see in the following sections that the circuits are quite robust to noise and amenable to efficient error mitigation techniques, since they use a small part of the entire Hilbert space. Third, we use a single type of two-qubit gate that is native or quasi-native to different hardware platforms. Fourth, we note that the connectivity of the circuit is quite local, where most of the qubits interact with very few qubits (for example 7/8 of the qubits need at most 4 neighboring connections) while the maximum number of interacting neighbors for any qubit is $\log d$. Here, we take advantage of the full connectivity of the IonQ hardware platform, so we can apply all gates directly. On a grid architecture, one would need to embed the circuit on the grid which asymptotically requires no more than doubling the number of qubits.

Next, we describe the distance estimation circuit. The power of the data loaders comes from the operations that one can do once the data is loaded into such "amplitude encoding" quantum states. In this work, we show how to use the data loader circuits to perform a fundamental operation at the core of supervised and unsupervised similarity-based learning, which is the estimation of the distance between data points.

Given two vectors x and y corresponding to classical data points, the Euclidean distance, namely $l_{xy} = \|x - y\|$[31] is given by

$$l_{xy} = \sqrt{\|x\|^2 + \|y\|^2 - 2\|x\|\|y\|c_{xy}}, \tag{4}$$

where $c_{xy} = \langle x|y\rangle$ is the inner product of the two normalized vectors. Here we describe a circuit to estimate c_{xy} which is combined with the classically calculated vector norms to obtain l_{xy}.

In fact, here we will only discuss one of the variants of the distance estimation circuits which works for the case where the inner product between the data points is positive, which is usually the case for image classification where the data points have all non negative coordinates. It is not hard to extend the circuit with one extra qubit to deal with the case of also non-positive inner products.

The distance estimation circuit for two data points of dimension d uses d qubits, $2(d-1)$ two-qubit parametrized gates, $2\log d$ depth, and allows us to measure at the end of the circuit a qubit whose probability of giving the outcome $|1\rangle$ is exactly the square of the inner product between the two normalized data points (see also Methods). From this, one can easily estimate the inner product and the distance between the original data points. The distance estimation circuit is explained in Fig. 1b.

We also notice a simplification we can make in the circuit that will reduce the number of gates and depth, as shown in Fig. 1c.

This reduces the number of gates of the circuit to $3d/2 - 2$ and reduces the depth by one. The final circuit used in our application is the one in Fig. 1c.

We now have all the necessary ingredients to implement the quantum Nearest Centroid classification circuit. As we have said, this is but a first, simple application of the above tools which can readily be used for other machine learning applications such as nearest neighbor classifiers or k-means clustering for unsupervised learning, where neural network techniques are not available. Let us start by briefly defining the Nearest Centroid algorithm in the classical setting. The first part of the algorithm is to use the training data to fit the model. This is a very simple operation of finding the average point of each class of data, meaning one adds all points with the same label and finds the "centroid" of each class. This part will be done classically and one can think of this cost as a one-time offline cost.

In the quantum case, one will still find the centroids classically and then pre-process each to find its norm and parameters for the gates of the data loader circuits. This does not change the asymptotic time of this step.

We now look at the second part of the Nearest Centroid algorithm which is the "predict" phase. Here, we want to assign a label to a number of test data points and for that we first estimate the distance between each data point and each centroid. For each data point we then assign the label of the centroid which is nearest to it.

The quantum Nearest Centroid is rather straightforward, it follows the steps of the classical algorithm apart from the fact that whenever one needs to estimate the distance between a data point and a centroid, we do this using the distance estimator circuit defined above.

One of the main advantages of using the Nearest Centroid as a basic benchmark for quantum machine learning is that we fully understand what the quantum algorithm does and how its runtime scales with the dimension of the data and the size of the data set. As described above, the circuit for estimating the distance of two d-dimensional data points has depth 2 log d. Theoretically, for an estimation of the distance up to accuracy ϵ one needs to run the circuit $N_s = O(1/\epsilon^2)$ times. In the future, amplitude estimation can be combined with this circuit in order to reduce the overall time to $O(\log(d)/\epsilon)$. The accuracy required depends on how well-classifiable our data set is, that is, whether most points are close to a centroid or they are mostly distributed equidistantly from the centroids. This number does not really depend on the dimension of the data set and in all data sets we considered, an approximation to the distance up to 0.1 for most points and 0.03 for a few difficult to classify points suffices, even for the MNIST dataset that originally has 784 dimensions. We argue that the number of shots will increase slowly as the problem sizes scale up. The Methods section discusses further strategies to optimize the number of shots.

Since the cost of calculating the circuit parameters is a one-off cost, if we want to estimate the distance between k centroids and n data points all of dimension d, then the quantum circuits will need d qubits and the running time would be of the form $O(kd + nd + kn\log(d)/\epsilon)$. The first term corresponds to pre-processing the centroids, the second term to pre-processing the new data points and the third term to estimating the distances between each data point and each centroid and assigning a label. The classical Nearest Centroid algorithm used in practice takes time $O(nkd)$, since for every of the n points one needs to compute the distance to each of k centroids, which takes time d. In theory, one can design quantum-inspired algorithms for Nearest Centroid and for other quantum machine learning applications that also use special classical data structures to approximate the distances[33]. These quantum-inspired algorithms depend poly-logarithmically in the dimension d but have much worse dependence on other parameters, including the rank of the data matrix and the error ϵ, which make them slower in practice than the classical ones currently used.

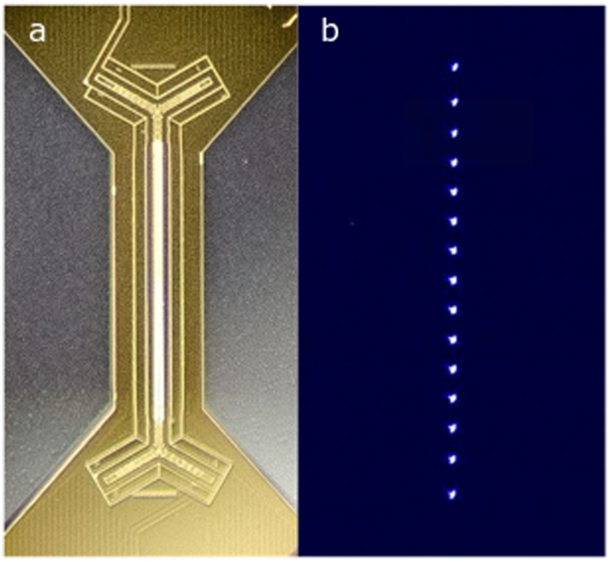


Fig. 2 Quantum processor. a A micro-fabricated surface electrode ion-trap, which is used to trap a linear chain of Ytterbium ions. **b** A chain of 15 ions is imaged by collecting fluorescence using an optical microscope, where each ion can represent a physical qubit.

Given the above runtime and scalability analysis, as well as the excellent accuracy that was achieved experimentally, we believe it is possible one can find parameter regimes where quantum computers, as they become bigger and faster, can provide an advantage for similarity-based classification. In particular, we would need the factor $\frac{\log d}{\epsilon}$ in the running time of the quantum algorithm to become smaller than d, or else the accuracy needed in these calculations to be not too high and the dimension to be large enough. In other words, we need $\epsilon \geq \frac{\log d}{d}$. As an example, in the experiments we performed on the MNIST dataset, an $\epsilon = 0.05$ is adequate for getting our high accuracy, which means that the dimension should be larger than 150.

Experiment

Our experimental demonstration is performed on an 11-qubit trapped ion processor based on $^{171}\mathrm{Yb}^{+}$ ion qubits. The device is commercially available through IonQ's cloud service. The 11-qubit device is operated with automated loading of a linear chain of ions (see Fig. 2), which is then optically initialized with high fidelity. Computations are performed using a mode-locked 355nm laser, which drives native single-qubit-gate (SQG) and two-qubit-gate (TQG) operations. The native TQG used is a maximally entangling Molmer Sorensen gate.

In order to maintain consistent gate performance, calibrations of the trapped ion processor are automated. Additionally, phase calibrations are performed for SQG and TQG sets, as required for implementing computations in queue and to ensure consistency of the gate perfomance.

The gates in the circuits were not performed in parallel in this experiment because this ability is not currently available through the cloud access to the IonQ quantum computers yet. However, such parallel application of gates have been demonstrated with high fidelity on trapped ion quantum computers at IonQ[32] and will be available in the near future.

We tested the algorithm with synthetic and real datasets. On an ideal quantum computer, the state right before measurement has non-zero amplitudes only for states within the unary basis,

Table 1. Comparison of labels assigned by the different classification schemes for synthetic data with $N_q = 4$, $N_c = 4$, and $N_s = 500$.

Sampling of points	0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3
Classical NC	0 0 2 0 0 0 0 0 1 0 1 1 1 1 1 1 1 1 3 1 2 2 2 2 2 2 3 2 2 2 3 3 3 3 3 3 3 3 3 3
QNC (no mitigation)	0 0 2 0 0 0 0 0 1 0 1 1 1 1 1 1 1 1 3 1 2 2 2 2 2 2 3 2 2 **1** 3 3 3 3 3 3 3 3 3 3
QNC (with mitigation)	0 0 2 0 0 0 0 0 1 0 1 1 1 1 1 1 1 1 3 1 2 2 2 2 2 2 3 2 2 **1** 3 3 3 3 3 3 3 3 3 3

The first line shows how the points were sampled, namely the first quarter of points were sampled starting from the centroid labeled 0, the second quarter were sampled from the centroid labeled 1, etc. The second row shows the labels assigned by the classical Nearest Centroid algorithm. These are the labels that we will benchmark against. The third and fourth line presents the results of the quantum Nearest Centroid algorithm without and with error mitigation. The labels in bold show where the quantum classification differs from the classical one.

Table 2. Comparison of labels assigned by the different classification schemes for synthetic data with $N_q = 8$, $N_c = 4$, and $N_s = 1000$.

Sampling of points	0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3
Classical NC	0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3
QNC (no mitigation)	0 0 0 0 0 0 0 0 0 0 **0 3 0 0 0 0 3 0 0 0** 2 2 2 2 2 2 2 2 2 2 3 3 3 **2** 3 3 3 3 **2** 3
QNC (with mitigation)	0 0 0 0 0 0 0 0 0 0 1 1 1 1 **0** 1 1 1 **0** 1 2 2 2 2 2 2 2 2 2 2 3 3 3 **2** 3 3 3 3 **2** 3

The labels in bold show where the quantum classification differs from the classical one.

$|e_i\rangle = |2^i\rangle$, i.e. states in which only a single bit is 1. The outcome of the experiment that is of interest to us is the probability of obtaining the state $|10..00\rangle$. On a real quantum computer, each circuit is run with a fixed number of shots to recreate the density matrix probability distribution in the output as closely as possible. Typically, there are many computational basis states other than the ones in the ideal probability distribution that are measured. Therefore, we have adopted two different techniques for estimating the desired probability.

1. No error-mitigation: Measure the first qubit and compute the probability as the ratio of the number of times the outcome is 1 over the total number of runs of the circuit N_s.
2. Error-mitigation: Measure all qubits and discard all runs of the circuit with results that are not of the form $|2^i\rangle$. Compute the probability as the ratio of the number of times the outcome is $|10..0\rangle$ over the total number of runs of the circuit where a state of the form $|2^i\rangle$ was measured.

Next we present our experimental results on synthetic data and the MNIST dataset. The Methods section also contains experimental results for the IRIS dataset.

For the synthetic data, we create datasets with N_c clusters of d-dimensional data points. This begins with generating k random points that serve as the "centroids" of a cluster, followed by generating $n = 10$ data points per centroid by adding Gaussian noise to the "centroid" point. The distance between the centroids was set to be a minimum of 0.3, the variance of the Gaussians was set to be 0.05 and the points were distributed within a sphere of radius 1. Therefore, there is appreciable probability of the points generated from one centroid lying closer to the centroid of another cluster. This can be seen as mimicking noise in real life data.

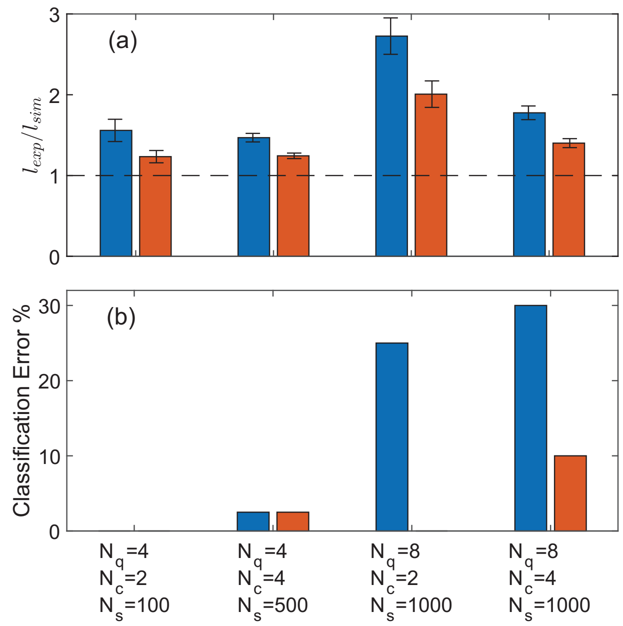


Fig. 3 Synthetic data. a Ratio of distance calculated from the quantum computer vs the simulator, and **b** classification of synthetic data for different number of clusters (N_c), qubits (N_q) and shots (N_s) before (blue, left) and after error mitigation (red, right). The number of data points was 10 per cluster. The classification error is calculated by comparing the quantum to the classical labels for each dataset. For the synthetic data, our goal was to see whether the quantum Nearest Centroid algorithm can assign the same labels as the classical one, even though the points are quite close to each other. Thus our classical benchmark for the synthetic data is 0% classification error. The baseline which corresponds to the accuracy of just randomly guessing the labels is $1/N_c$. The error bars in **a** correspond to the standard error in the ratio of distances for each dataset.

In Tables 1 and 2 we present, as an example, the data points and the labels assigned to them by the classical and quantum Nearest Centroid for the case of four classes in the 4- and 8-dimensional case. We see in these examples, and in fact this is the behavior overall, that our quantum Nearest Centroid algorithm with error mitigation provides the correct labels for 39 out of 40 4-dimensional points and for 36 out of the 40 8-dimensional points, or an accuracy of 97.5% and 90% respectively. The quantum classification is affected by the noise in the operations and can shift the assignment non-trivially based on the accuracy in the measured distance, as can be seen in cluster 1 in Table 2. This will be discussed in more detail later in this section.

In Fig. 3b, we first show the error of the quantum Nearest Centroid algorithm averaged over a data set of 20 or 40 data points. Here, N_c is the number of classes, N_s is the number of shots and $N_q = d$ is the dimension and number of qubits used in the experiment. For each case, there are two values (left-blue, right-red) that correspond to the results without and with error mitigation. For 2 classes, we see that for the case of 4-dimensional data, we achieve 100% percent accuracy even with as little as 100 shots per circuit and even without error-mitigation.

For the case of 8-dimensional data, we also achieve 100% accuracy with error mitigation and 1000 shots. We also performed experiments with four classes, and the accuracies were 97.5% for the 4-dimensional case and 90% for the 8-dimensional case with

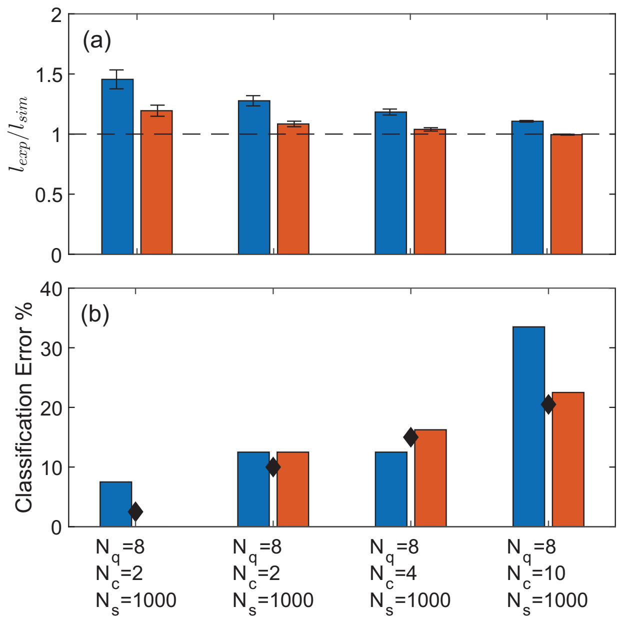


Fig. 4 MNIST data set. (**a**) Ratio of distance calculated from experiment vs simulator, and (**b**) classification of different down-scaled MNIST data sets. From left to right, the data sets were (i) 0 and 1 (40 samples), (ii) 2 and 7 (40 samples), (iii) 0–3 (80 samples) and (iv) 0–9 (200 samples). The diamonds show the accuracy of classical Nearest Centroid. The error bars in (a) correspond to the standard error in the ratio of distances for each dataset.

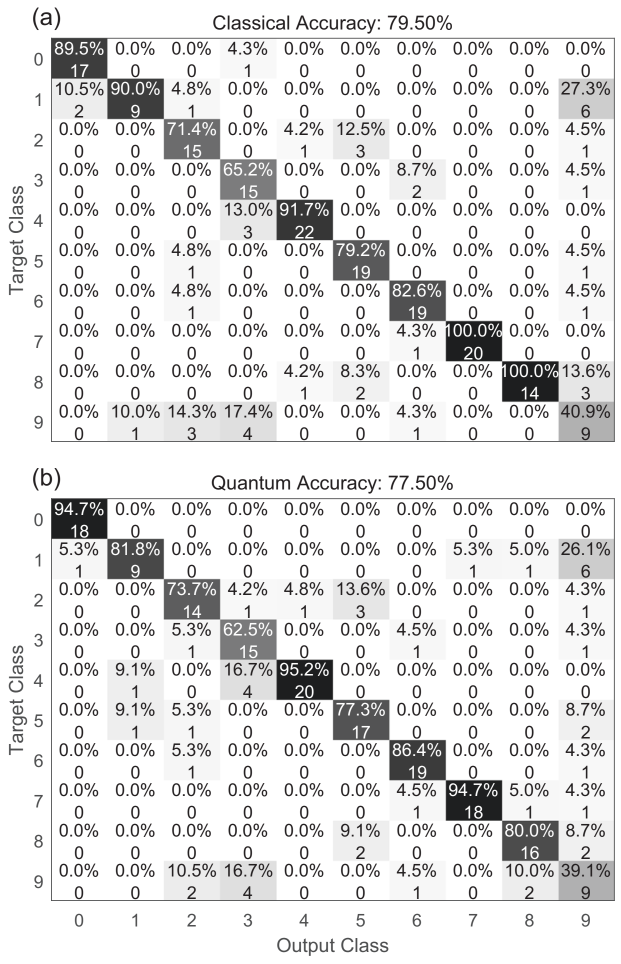


Fig. 5 Confusion matrices. Confusion matrices for MNIST classification **a** from classical, and **b** quantum computer with 10 classes and 200 points after error mitigation.

error-mitigation. The number of shots used is more for the higher dimensional case as suggested by the analysis later in this Section.

Figure 3a shows the average ratio between the distance estimation from experiment and simulation. We see that the experimentally estimated distance may actually be off by a significant factor from the theoretical distance. However, since the error bars in the distance estimation are quite small, and what matters for classification is that the ratio of the distances remains accurate, it is reasonable that we have high classification accuracy. In the Methods section, we discuss a way to reduce the error in the distance itself based on knowledge of the noise model of the quantum system.

Next, we discuss our results for the MNIST dataset. The MNIST database contains 60,000 training images and 10,000 test images of handwritten digits and it is widely used as a benchmark for classification. Each image is a 28×28 image i.e., a 784-dimensional point. To work with this data set we preprocessed the images with PCA to project them into 8 dimensions. This reduces the accuracy of the algorithms, both the classical and the quantum, but it allows us to objectively benchmark the quantum algorithms and hardware on different types of data.

For our experiments with the 8-dimensional MNIST database, we created four different data sets. The first one has 40 samples of 0 and 1 digits only. The second one has 40 samples of the 2 and 7 digits. The third one has 80 samples of four different digits 0–3. The fourth one has 200 samples of all possible digits 0–9.

In Fig. 4 we see that for the first data set the quantum algorithm with error mitigation gets accuracy 100%, matching well the classical accuracy of 97.5%. For the case of 2 vs. 7, we achieve accuracy of around 87.5% similar to the classical algorithm. For the four-class classification, we also get similar accuracy to the classical algorithm of around 83.75%. Most impressively, for the last dataset with all 10 different digits, we practically match the classical accuracy of around 77.5% with error mitigation and 1000 shots. We also provide the ratio of the experimental vs simulated distance estimation for these experiments which shows high accuracy for the distance estimation as well.

In Fig. 5 we provide the confusion graph for the classical and quantum nearest centroid algorithm, showing how our quantum algorithm matches the accuracy of the classical one.

In this section, we discuss the different sources of error and their scaling. One part of the effective error in the circuit is coherent and changes the state within the unary encoding subspace during two-qubit operations. This can be represented for each two qubit operation as $RBS(\theta) \to RBS(\theta(1+\Gamma r))$, where Γ is the magnitude of the noise and r is a normally distributed random number. For simplicity of the calculations we can simply say that each $RBS(\theta)$ gate performs a linear mapping for a vector (a_1, a_2) in the basis $\{e_1, e_2\}$ of the form:

$$(a_1, a_2) \mapsto (\cos\theta_\Gamma \cdot a_1 + \sin\theta_\Gamma \cdot a_2, -\sin\theta_\Gamma \cdot a_1 + \cos\theta_\Gamma \cdot a_2) \tag{5}$$

such that $\cos\theta - \cos\theta_\Gamma \approx \Gamma r$ and $\sin\theta - \sin\theta_\Gamma \approx -\Gamma r$.

It is not hard to calculate now that each layer of the circuit only adds an error of at most $\sqrt{2}\Gamma$, even though one layer can consist of up to $n/2RBS$ gates. To see this, consider any layer of the distance estimation circuit, for example, the layer on time step 3 in Fig. 1. Let the state before this layer be denoted by a unit vector $a = (a_1, a_2, \ldots, a_n)$ in the basis $\{e_1, e_2, \ldots, e_n\}$. Then, if we look at the difference of the output of the layer when applying the correct

RBS gates and the ones with Γ error, we get an error vector of the form

$$e_\Gamma = (\Gamma r_1 a_1 + \Gamma r_1 a_2, -\Gamma r_1 a_1 + \Gamma r_1 a_2, \dots) \tag{6}$$

If we look at the ℓ_2 norm of this vector we have

$$\|e_\Gamma\|^2 \leq 2\Gamma^2 \|a\|^2 = 2\Gamma^2 \tag{7}$$

Since the number of layers is logarithmic in the number of qubits, this implies that the overall accuracy to leading order at the end is $(1-\Gamma)^{O(\log(n))}$. This is one of the most interesting characteristics of our quantum circuits, whose architecture and shallow depth enables accurate calculations even with noisy gates. This implies that, for example, for 1024-dimensional data and a desired error in the distance estimation of 10^{-2} we need the fidelity of the TQG gates to be of the order of 10^{-3} and not 10^{-5}, as would be the case if the error grew with the number of gates.

Next, we also expect some level of depolarizing error in the experiment. Measurements in the computational basis can be modeled by a general output density matrix of the form

$$\rho_d = p \sum_{i=0}^{n} |a_{i\Gamma}|^2 |e_i\rangle\langle e_i| + \frac{(1-p)}{2^n} I_{2^n}, \tag{8}$$

where $a_{i\Gamma}$ are the amplitudes of the quantum state after the error described previously is incorporated. The depolarizing error can be mitigated by discarding the histogram states that result in states that are not $|e_i\rangle$. After this post-selection, the resulting density matrix is

$$\rho = \frac{1}{\mathcal{N}} \sum_{i=1}^{n} \left(p|a_{i\Gamma}|^2 + \frac{(1-p)}{2^n} \right) |e_i\rangle\langle e_i| \tag{9}$$

Here $p = f^m$, where f is the fidelity of two-qubit gates and m is the number of two-qubit gates. For our circuits, $m = 4.5n - 6$. The normalization factor, $\mathcal{N} = \sum_{i=1}^{n} \left(p|a_{i\Gamma}|^2 + \frac{(1-p)}{2^n} \right)$. Therefore, the vector overlap measured from the post-selected density matrix is

$$\begin{aligned} c &= \frac{p|a_{1\Gamma}|^2 + \frac{(1-p)}{2^n}}{\sum_{i=1}^{n} \left(p|a_{i\Gamma}|^2 + \frac{(1-p)}{2^n} \right)} \\ &= \frac{|a_{1\Gamma}|^2 + \frac{(1-p)}{2^n p}}{\sum_{i=1}^{n} \left(|a_{i\Gamma}|^2 + \frac{(1-p)}{2^n p} \right)} \\ &= \frac{|a_{1\Gamma}|^2 + \frac{(1-p)}{2^n p}}{1 + \frac{(1-p)n}{2^n p}} \end{aligned} \tag{10}$$

For effective error mitigation, we need $2^n p \gg 1 \Rightarrow 2^n f^{4.5n-6} \gg 1$. When $n \gg 1$, this gives $2f^{4.5} \gg 1 \;\Rightarrow\; f \gg \frac{1}{2^{1/4.5}} = 85.8\%$. Thus, we find a threshold for the fidelity over which $c \to |a_{1\Gamma}|^2$ as n increases. Since the depolarizing error is much lower than this value, this implies post selection will become more effective at removing depolarizing error as the problem size increases. We validate this error model by comparing against experimental data in the "Methods".

The number of samples needed to get sufficient samples from the ideal density matrix after performing the post-selection to remove depolarizing error increases exponentially with the number of qubits n. However, with the coming generation of ion trap quantum computers, this error will be low and thus the coefficient of the exponential increase will be extremely small[34]. The main source of remaining error will be the first one that perturbs the state within the encoding subspace but this grows only logarithmically with n.

DISCUSSION

We presented an implementation of an end-to-end quantum application in machine learning. We performed classification of synthetic and real data using QCWare's quantum Nearest Centroid algorithm and IonQ's 11-qubit quantum hardware. The results are extremely promising with accuracies that reach 100% for many cases with synthetic data and match classical accuracies for the real data sets. In particular, we note that we managed to perform a classification between all 10 different digits of an MNIST data set down-scaled to 8-dimensions with accuracy matching the classical nearest centroid one for the same data set. To our knowledge such results have not been previously achieved on any quantum hardware and using any quantum classification algorithm.

We also argue for the scalability of our approach and its applicability for NISQ machines, based on the fact that the algorithm uses shallow quantum circuits that are provably tolerant to certain levels of noise. Further, the particular application of classification is amenable to computations with limited accuracy since it only needs a comparison of distances between different centroids but not finding all distances to high accuracy.

For our experiments, we used unary encoding for the data points. While this increases the number of qubits needed to the same as in quantum variational methods, it allows for removal of depolarizing error, and coherent error that grows only logarithmically in the number of qubits. This allows us to be optimistic for applying similar data loading techniques to the next generation of quantum computers as an input to much more classically complicated procedures including training neural networks or performing Principal Component or Linear Discriminant Analysis. With 99.9% fidelity of IonQ's newest 32 qubit system, we expect to continue to match or outperform classical accuracy in machine learning on higher-dimensional datasets without error correction.

The question of whether quantum machine learning can provide real world applications is still wide open but we believe our work brings the prospect closer to reality by showing how joint development of algorithms and hardware can push the state-of-the-art of what is possible on quantum computers.

METHODS

Previous proposals for loading classical data into quantum states

There have been several proposals for acquiring fast quantum access to classical data that loosely go under the name of Quantum Random Access Memory (QRAM). A QRAM, as described in[35,36], in some sense would be a specific hardware device that could "natively" access classical data in superposition, thus having the ability to create quantum states like the one defined above in logarithmic time. Given the fact that such specialized hardware devices do not yet exist, nor do they seem to be easy to implement, there have been proposals for using quantum circuits to perform similar operations. For example, a circuit to perform the bucket brigade architecture was defined in[37], where a circuit with $O(d)$ qubits and $O(d)$ depth was described and also proven to be robust up to a level of noise. A more "brute force" way of loading a d-dimensional classical data point is through a multiplexer-type circuit, where one can use only $O(\log d)$ qubits but for each data point one needs to sequentially apply d log d-qubit-controlled gates, which makes it quite impractical. Another direction is loading classical data using a unary encoding. This was used in[38] to describe finance applications, where the circuit used $O(d)$ qubits and had $O(d)$ depth. A parallel circuit for specifically creating the W state also appeared in[39].

Method for finding the gate parameters

We describe here the procedure that given access to a classical data point $x = (x_1, x_2, \dots, x_d) \in \mathbb{R}^d$, pre-processes the classical data efficiently, i.e., spending only $\tilde{O}(d)$ total time, in order to create a set of parameters $\theta = (\theta_1, \theta_2, \dots, \theta_{d-1}) \in \mathbb{R}^{d-1}$, that will be the parameters of the $(d-1)$ two-qubit gates we will use in our quantum circuit.

The data structure for storing θ is similar to the one used in[14], but note that[14] assumed quantum access to these parameters (in the sense of being able to query these parameters in superposition). Here, we will compute and store these parameters in the classical FPGA controller of the quantum circuit to be used as the parameters of the gates in the quantum circuit.

At a high level, we think of the coordinates x_i as the leaves of a binary tree of depth log d. The parameters θ correspond to the values of the internal tree nodes, starting form the root and going towards the leaves.

We first consider the parameter series $(r_1, r_2, \ldots, r_{d-1})$. For the last $d/2$ values $(r_{d/2}, \ldots, r_{d-1})$, we define an index j that takes values in the interval $[1, d/2]$ and define the values as

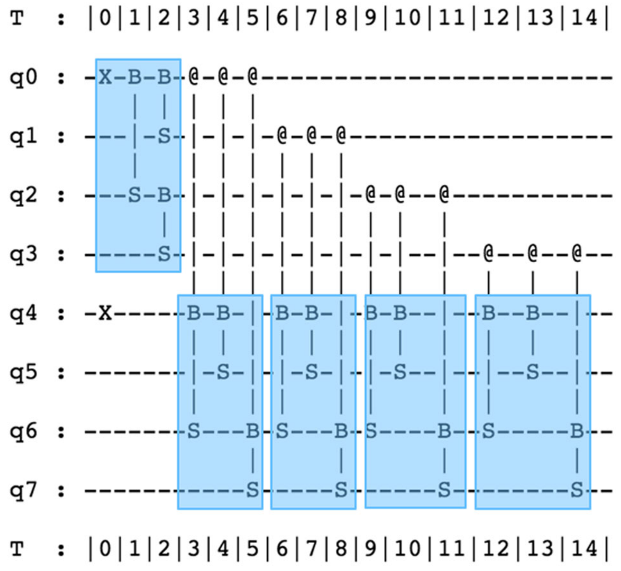
$$r_{d/2+j-1} = \sqrt{x_{2j}^2 + x_{2j-1}^2} \tag{11}$$

For the first $d/2 - 1$ values, namely the values of $(r_1, r_2, \ldots, r_{d/2-1})$, and for j in $[1, d/2]$, we define

$$r_j = \sqrt{r_{2j+1}^2 + r_{2j}^2} \tag{12}$$

We can now define the set of angles $\theta = (\theta_1, \theta_2, \ldots, \theta_{d-1})$ in the following way. We start by defining the last $d/2$ values $(\theta_{d/2}, \ldots, \theta_{d-1})$. To do so, we define an index j that takes values in the interval $[1, d/2]$ and define the values as

$$\theta_{d/2+j-1} = \arccos\left(\frac{x_{2j-1}}{r_{d/2+j-1}}\right), \text{ if } x_{2j} \text{ is positive}$$
$$\theta_{d/2+j-1} = 2\pi - \arccos\left(\frac{x_{2j-1}}{r_{d/2+j-1}}\right), \text{ if } x_{2j} \text{ is negative.}$$



```
T  : |0|1|2|3|4|5|6|7|8|9|10|11|12|13|14|

q0 : -X-B-B-@-@-@------------------------
        | | | | |
q1 : ---|-S-|-|-|-@-@-@------------------
        |   | | | | | |
q2 : ---S-B-|-|-|-|-|-|-@-@--@-----------
          | | | | | | | | |  |
q3 : -----S-|-|-|-|-|-|-|-|--|--@--@--@--
            | | | | | | | |  |  |  |  |
q4 : -X-----B-B-|-B-B-|-B-B--|--B--B--|--
            | | | | | | | |  |  |  |  |
q5 : -------|-S-|-|-S-|-|-S--|--|--S--|--
            |   | |   | |    |  |     |
q6 : -------S---B-S---B-S----B--S-----B--
                |     |      |        |
q7 : -----------S-----S------S--------S--

T  : |0|1|2|3|4|5|6|7|8|9|10|11|12|13|14|
```

Fig. 6 An optimized loader for a 16-dimensional data point, seen as a 4 × 4 matrix. The blue boxes correspond to the parallel loader from Fig. 1 and its controlled versions.

For the first $d/2 - 1$ values, namely the values for $j \in [1, d/2]$, we define

$$\theta_j = \arccos\left(\frac{r_{2j}}{r_j}\right) \tag{13}$$

Note that we can easily perform these calculations in a read-once way, where for every x_i we update the values that are on the path from the i-th leaf to the root. This also implies that when one coordinate of the data is updated, then the time to update the θ parameters is only logarithmic, since only log d values of r and of θ need to be updated.

Method for an optimized data loader

An interesting extension of the parallel data loader we defined above is that we can trade off qubits with depth and keep the number of overall gates $d - 1$ (in this case, both *RBS* and controlled-*RBS* gates).

For example, we can use $2\sqrt{d}$ qubits and $O(\sqrt{d}\log d)$ depth[31]. The circuit is quite simple, if one thinks of the d dimensional vector as a $\sqrt{d} \times \sqrt{d}$ matrix. Then, we can index the coordinates of the vector using two registers (one each for the row and column) and create the state

$$|x\rangle = \frac{1}{\|x\|}\sum_{i,j=1}^{\sqrt{d}} x_{ij}|e_i\rangle|e_j\rangle \tag{14}$$

For this circuit, we find, in the same way as for the parallel loader, the values θ and then create the following circuit in Fig. 6. We start with a parallel loader for a $\sqrt{d}$-dimensional vector using the first $\sqrt{d}$ angles θ (which corresponds to a vector of the norms of the rows of the matrix) and then, controlled on each of the $\sqrt{d}$ qubits we perform a controlled parallel loader corresponding to each row of the matrix.

Notice that naively the depth of the circuit is $O(d\log d)$, but it is easy to see that one can interleave the gates of the controlled-parallel loaders to an overall depth of $O(\sqrt{d}\log d)$. We will not use this circuit here but such circuits can be useful both for loading vectors and in particular matrices for linear algebraic computations.

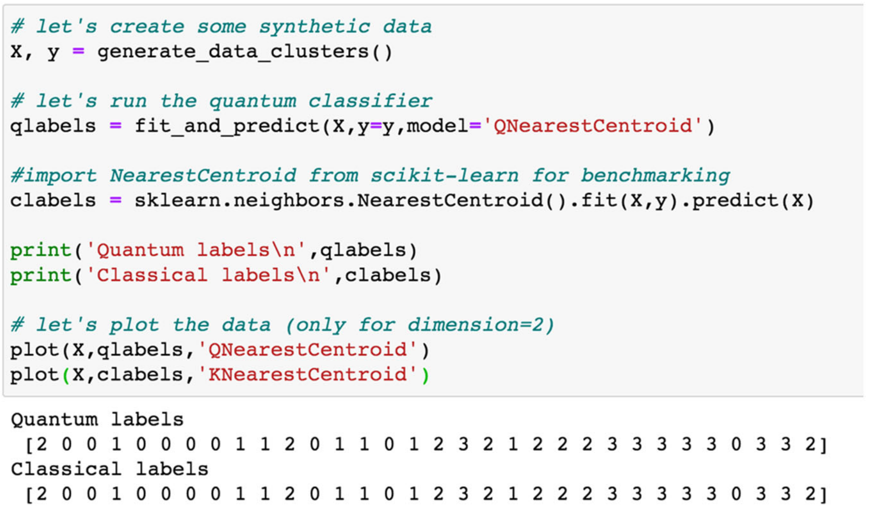
Method for estimating the distance using the distance estimation circuit

The distance estimation circuit is explained in Fig. 1. Here we see how from this circuit we can estimate the square of the inner product between the two vectors.

After the first part of the circuit, namely the loader for the vector x (time steps 0–3), the state of the circuit is $|x\rangle$, as in Eq. (3). One can rewrite this state in the basis $\{|y\rangle, |y^\perp\rangle\}$ as

$$\langle x, y\rangle|y\rangle + \sqrt{1 - |\langle x, y\rangle|^2}|y^\perp\rangle \tag{15}$$

Once the state goes through the inverse loader circuit for y the first part of the superposition gets transformed into the state $|e_1\rangle$ (which would go to the state $|0\rangle$ after an X gate on the first qubit), and the second part of the



```
# let's create some synthetic data
X, y = generate_data_clusters()

# let's run the quantum classifier
qlabels = fit_and_predict(X,y=y,model='QNearestCentroid')

#import NearestCentroid from scikit-learn for benchmarking
clabels = sklearn.neighbors.NearestCentroid().fit(X,y).predict(X)

print('Quantum labels\n',qlabels)
print('Classical labels\n',clabels)

# let's plot the data (only for dimension=2)
plot(X,qlabels,'QNearestCentroid')
plot(X,clabels,'KNearestCentroid')
```

```
Quantum labels
 [2 0 0 1 0 0 0 0 1 1 2 0 1 1 0 1 2 3 2 1 2 2 2 3 3 3 3 3 0 3 3 2]
Classical labels
 [2 0 0 1 0 0 0 0 1 1 2 0 1 1 0 1 2 3 2 1 2 2 2 3 3 3 3 3 0 3 3 2]
```

Fig. 7 Code. An example of a jupyter notebook that runs the Quantum Nearest Centroid algorithm, as well as the scikit-learn Nearest Centroid algorithm, and prints and plots the results.

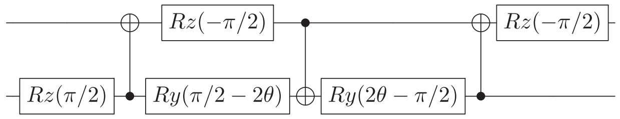


Fig. 8 Gate decomposition. Circuit to implement $iRBS(\theta)$ gate.

superposition goes to a superposition of states $|e_j\rangle$ orthogonal to $|e_1\rangle$

$$\langle x,y\rangle|e_1\rangle+\sqrt{1-|\langle x,y\rangle|^2}|e_1^{\perp}\rangle \tag{16}$$

It is now clear that after measuring the circuit (either all qubits or just the first qubit), the probability of getting $|1\rangle$ in the first qubit is exactly the square of the inner product of the two data points.

Software

The development of the quantum software followed one of the most popular classical ML libraries, called scikit-learn (https://scikit-learn.org/), where a classical version of the Nearest Centroid algorithm is available. In the code snippet in Fig. 7 we can see how one can call the quantum and classical Nearest Centroid algorithm with synthetic data (one could also use user-defined data) through QCWare's platform Forge, in a jupyter notebook.

The function "fit-and-predict" first classically fits the model. For the prediction it calls a function "distance-estimation" for each centroid and each data point. The "distance-estimation" function runs the procedure we described above, using the function "loader" for each input and returns an estimate of the Euclidean distance between each centroid and each data point. The label of each data point is assigned as the label of the nearest centroid.

Note that one could imagine more quantum ways to perform the classification, where, for example, instead of estimating the distance for each centroid separately, this operation could happen in superposition. This would make the quantum algorithm faster but also increase the number of qubits needed. We remark also that the distance or inner product estimation procedure can find many more applications such as matrix-vector multiplications during clustering or training neural networks.

Implementation of the circuits on the IonQ processor

The circuits we described above are built with the $RBS(\theta)$ gates (Eq. (2)). To map these gates optimally onto the hardware we will instead use the modified gate

$$iRBS(\theta)=\begin{pmatrix} IIII1 & 0 & 0 & 0\\ 0 & \cos\theta & -i\sin\theta & 0\\ 0 & -i\sin\theta & \cos\theta & 0\\ 0 & 0 & 0 & 1\end{pmatrix} \tag{17}$$

It is easy to see that the distance estimation circuit stays unchanged. Using the fact that $iRBS(\theta)=\exp(i\theta(\sigma_x\otimes\sigma_x+\sigma_y\otimes\sigma_y))$, we can decompose the gate into the circuit shown in Fig. 8[40]. Each CNOT gate can be implemented with a maximally entangling Molmer Sorensen gate and single qubit rotations native to IonQ hardware.

We run 4 and 8 qubit versions of the algorithm. The 4 qubit circuits have 12 TQG and the 8 qubit circuits have 30 TQG.

Validating the noise model

To test the noise model, we notice that the experimentally measured value of the overlap, $c_{\exp}$ should be proportional to $|a_{1\Gamma}|^2\propto c_{\mathrm{sim}}$. We plot $c_{\exp}$ vs c_{sim} in Fig. 9 for the synthetic dataset with $N_q=8$, $N_c=4$ and $N_s=1000$ and find that the data fits well to straight lines. Using Eq. (9), we know that the slope of the line before error mitigation should be f^m. From this, we can estimate the value for f as 95.85% which is remarkably close to the expected two qubit gate fidelity of 96%.

The fact that the data can be fit to a straight line can be straightforwardly used to obtain a better estimate of the distance. In this work, we have focused on classification accuracy which is robust to errors in the distance as long as they occur in the distances measured to all centroids. Nevertheless, there may be applications in which the distance (or the inner product) needs to be measured accurately, for example when we want to perform matrix vector multiplications. Note that classification is based on which centroid is nearest to the data point and thus if all distances are corrected in the same way, this will not change the classification label.

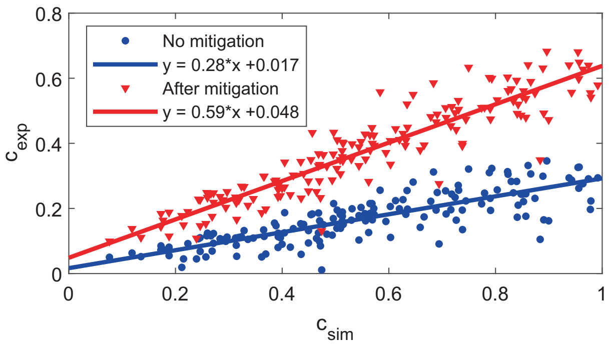


Fig. 9 The experimentally estimated value of c vs its value from simulation. This data correspond to the synthetic dataset with $N_q=4$, $N_d=8$, and $N_s=1000$.

Reducing statistical error

In future experiments with much higher fidelity gates, the approximation to the distance will become better with the number of runs and scale as $1/\sqrt{N_S}$. Increasing the number of measurements is important for those points that are almost equidistant between one or more clusters. One could imagine an adaptive schedule of runs where initially a smaller number of runs is performed with the same data point and each centroid and then depending on whether the nearest centroid can be clearly inferred or not, more runs are performed with respect to the centroids that were near the point. We haven't performed such optimizations here since the number of runs remains very small. As we have said, with the advent of the next generation of quantum hardware, one can apply amplitude estimation procedures to reduce the number of samples needed.

The IRIS dataset

Here we discuss an additional experiment that was performed on the quantum computer. The IRIS data set consists of three classes of flowers (Iris setosa, Iris virginica and Iris versicolor) and 50 samples from each of three classes. Each data point has four dimensions that correspond to the length and the width of the sepals and petals, in centimeters. This data set has been used extensively for benchmarking classification techniques in machine learning, in particular because classifying the set is non trivial. In fact, the data points can be clustered easily in two clusters, one of the clusters contains Iris setosa, while the other cluster contains both Iris virginica and Iris versicolor. Thus classification techniques like Nearest Centroid do not work exceptionally well without preprocessing (for example linear discriminant analysis) and hence it is a good data set to benchmark our quantum Nearest Centroid algorithm as it is not tailor-made for the method to work well.

Figure 10 shows the classification error for the IRIS data set of 150 4-dimensional data points. The classical Nearest Centroid classifies around 92.7% of the points while our experiments with 500 shots and error mitigation reaches 84% accuracy. Increasing the number of shots beyond 500 does not increase the accuracy because at this point the experiment is dominated by systematic noise which changes each time the system is calibrated. In the particular run, going from 500 to 1000 shots, the number of wrong classifications slightly increases, which just reflects the variability in the calibration of the system. We also provide the ratio of the experimental vs simulated distance estimation for these experiments.

Figure 10c, d compares the classification visually before and after error mitigation. Before error mitigation, many of the points that lie close to midway between centroids are mis-classified, whereas applying the mitigation moves them to the right class.

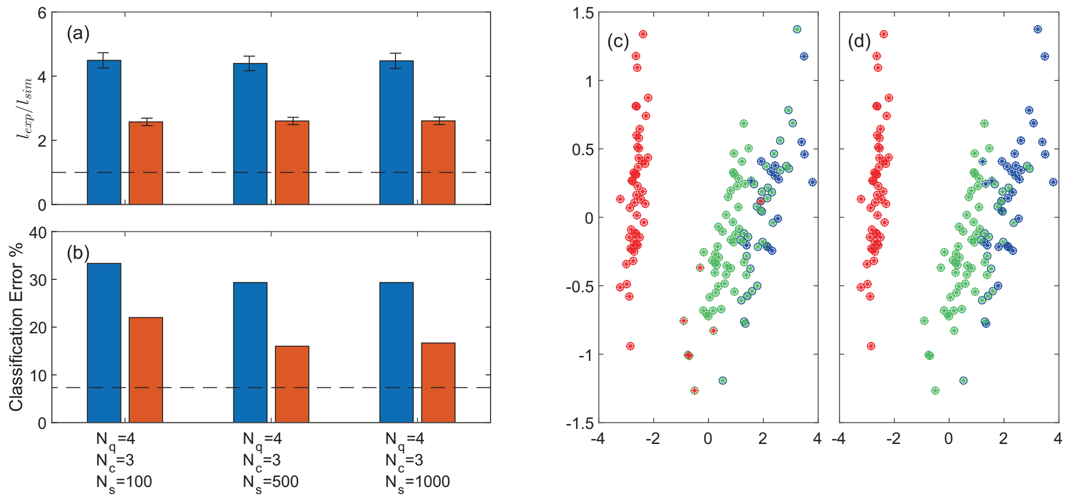


Fig. 10 **Iris data set. a** Ratio of distance calculated from the quantum computer vs the simulator, and **b** classification of Iris data. There were 150 data points in total. The dashed line shows the accuracy of classical nearest centroid algorithm. The error bars in **a** correspond to the standard error in the ratio of distances for each dataset. Iris data classification pictured after using principal component analysis to reduce the dimension from 4 to 2. The color of the boundary of the circles indicates the three human-assigned labels. The color of the interior indicates the class assigned by the quantum computer. **c** shows the classification before error mitigation using 500 shots, and **d** shows the classification after.

DATA AVAILABILITY

The data supporting the study's findings are available from the corresponding author upon reasonable request.

Received: 5 January 2021; Accepted: 1 July 2021;
Published online: 05 August 2021

REFERENCES

1. Debnath, S. et al. Demonstration of a small programmable quantum computer with atomic qubits. *Nature* **536**, 63–66 (2016).
2. Monz, T. et al. Realization of a scalable shor algorithm. *Science* **351**, 1068–1070 (2016).
3. Barends, R. et al. Superconducting quantum circuits at the surface code threshold for fault tolerance. *Nature* **508**, 500–503 (2014).
4. Córcoles, A. D. et al. Demonstration of a quantum error detection code using a square lattice of four superconducting qubits. *Nat. Commun.* **6**, 6979 (2015).
5. Linke, N. M. et al. Experimental comparison of two quantum computing architectures. *Proc. Natl Acad. Sci.* **114**, 3305–3310 (2017).
6. Murali, P. et al. Full-stack, real-system quantum computer studies: architectural comparisons and design insights. In *Proceedings of the 46th International Symposium on Computer Architecture*, ISCA '19, 527 (Association for Computing Machinery, New York, NY, USA, 2019). https://doi.org/10.1145/3307650.3322273.
7. Arute, F. et al. Quantum supremacy using a programmable superconducting processor. *Nature* **574**, 505–510 (2019).
8. Suzuki, Y. et al. Amplitude estimation without phase estimation. *Quantum Inform. Process.* (2020).
9. Tanaka, T. et al. Amplitude estimation via maximum likelihood on noisy quantum computer. *arXiv:2006.16223* (2020).
10. Grinko, D., Gacon, J., Zoufal, C. & Woerner, S. Iterative quantum amplitude estimation. *npj Quantum Inf.* **7**, 52 (2021).
11. Aaronson, S. & Rall, P. Quantum Approximate Counting, simplified. In *Proceedings of SIAM Symposium on Simplicity in Algorithms* (2020).
12. Bouland, A., van Dam, W., Joorati, H., Kerenidis, I. & Prakash, A. Prospects and challenges of quantum finance. *arXiv:2011.06492* (2020).
13. Lloyd, S., Mohseni, M. & Rebentrost, P. Quantum principal component analysis. *Nat. Phys.* **10**, 631–633 (2014).
14. Kerenidis, I. & Prakash, A. Quantum Recommendation Systems **67**,49:1–49:21 (2017).
15. Kerenidis, I., Landman, J., Luongo, A. & Prakash, A. q-means: A quantum algorithm for unsupervised machine learning. In Wallach, H. et al. (eds.) *Advances in Neural Information Processing Systems* **32**, 4136-4146 (Curran Associates, Inc., 2019).
16. Li, T., Chakrabarti, S. & Wu, X. Sublinear quantum algorithms for training linear and kernel-based classifiers. In Chaudhuri, K. & Salakhutdinov, R. (eds.) *Proceedings of the 36th International Conference on Machine Learning* of *Proceedings of Machine Learning Research*, **97**, 3815–3824 (PMLR, Long Beach, California, USA, 2019). http://proceedings.mlr.press/v97/li19b.html.
17. Noh, H., You, T., Mun, J. & Han, B. Regularizing deep neural networks by noise: Its interpretation and optimization. *NeurIPS* (2017).
18. Tang, E. A quantum-inspired classical algorithm for recommendation systems. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing - STOC 2019* (ACM Press, 2019). https://doi.org/10.1145/3313276.3316310.
19. Farhi, E. & Neven, H. Classification with quantum neural networks on near term processors. *arXiv:1802.06002* (2018).
20. Cong, I., Choi, S. & Lukin, M. D. Quantum convolutional neural networks. *Nat. Phys. 15* (2019).
21. Cerezo, M., Sone, A., Volkoff, T., Cincio, L. & Coles, P. J. Cost-function-dependent barren plateaus in shallow quantum neural networks. *Nat. Commun.* **12***,* 1791 (2021).
22. Cerezo, M., Sone, A., Cincio, L. & Coles, P. Barren plateau issues for variational quantum-classical algorithms. *Bull. Am. Phys. Soc.* **65**, (2020).
23. Sharma, K., Cerezo, M., Cincio, L. & Coles, P. Trainability of dissipative perceptron-based quantum neural networks. *arXiv preprint arXiv:2005.12458* (2020).
24. Garcıa-Hernandez, H. I., Torres-Ruiz, R. & Sun, G.-H. Image classification via quantum machine learning. *arXiv:2011.02831* (2020).
25. Nakaji, K. & Yamamoto, N. Quantum semi-supervised generative adversarial network for enhanced data classification. *arXiv:2010.13727* (2020).
26. Cappelletti, W., Erbanni, R. & Keller, J. Polyadic quantum classifier. *arXiv:2007.14044* (2020).
27. Kumar, S., Dangwal, S. & Bhowmik, D. Supervised learning using a dressed quantum network with "super compressed encoding": Algorithm and quantum-hardware-based implementation. *arXiv:2007.10242* (2020).
28. Havlicek, V. et al. Supervised learning with quantum enhanced feature spaces. *Nature.* **567**, 209–212 (2019).
29. Grant, E. et al. Hierarchical quantum classifiers. *npj Quantum Inf.* **4**, 65 (2018).
30. et al. M. B. Quantum-assisted learning of hardware-embedded probabilistic graphical models. *Phys. Rev. X* **7**, 041052 (2017).
31. Kerenidis, I. A method for loading classical data into quantum states for applications in machine learning and optimization. *U.S. Patent Application No. 16/986,553 and 16/987,235* (2020).
32. N. Grzesiak, K. W. e. a., R. Blümel. Efficient arbitrary simultaneously entangling gates on a trapped-ion quantum computer. *Nat. Commun.* **11,** 2963 (2020).
33. Chia, N. et al. Sampling-based sublinear low-rank matrix arithmetic framework for dequantizing quantum machine learning. In: *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing* (2020).
34. Wang, Y. et al. High-fidelity two-qubit gates using a microelectromechanical-system-based beam steering system for individual qubit addressing. *Phys. Rev. Lett.* **125**, 150505 (2020).
35. Giovannetti, V., Lloyd, S. & Maccone, L. Architectures for a quantum random access memory. *Phys. Rev. A* **78**, 052310 (2008).
36. Giovannetti, V., Lloyd, S. & Maccone, L. Quantum random access memory. *Phys. Rev. Lett.* **100**, 160501 (2008).

37. Arunachalam, S., Gheorghiu, V., Jochym-O'Connor, T., Mosca, M. & Srinivasan, P. V. On the robustness of bucket brigade quantum ram. *New Journal of Physics* **17**, 123010 (2020).
38. Ramos-Calderer, S. et al. Quantum unary approach to option pricing. *Phys. Rev. A* **103**, 032414 (2021).
39. Cruz, D. et al. Efficient quantum algorithms for ghz and w states, and implementation on the ibm quantum computer. *Adv. Quantum Technol.* 1900015 (2019).
40. Vatan, F. & Williams, C. Optimal quantum circuits for general two-qubit gates. *Phys. Rev. A* **69**, 032315 (2004).

ACKNOWLEDGEMENTS

This work is a collaboration between QCWare and IonQ. We acknowledge helpful discussions with Peter McMahon.

AUTHOR CONTRIBUTIONS

S.J. compiled the quantum circuits, did the data analysis and designed the error mitigation technique. S.D. was responsible for the quantum hardware performance. A.M. and A.S. contributed to the quantum software development. A.P. contributed to the design of the quantum algorithms. J.K. initiated the project and contributed to the hardware. I.K. conceived the project and contributed in all aspects.

COMPETING INTERESTS

The authors declare no competing interests.

ADDITIONAL INFORMATION

Correspondence and requests for materials should be addressed to S.J.

Reprints and permission information is available at http://www.nature.com/reprints