BIG DATA MINING AND ANALYTICS
ISSN 2096-0654 05/06 pp 56–67
Volume 3, Number 1, March 2020
DOI: 10.26599/BDMA.2019.9020018

On Quantum Methods for Machine Learning Problems Part II: Quantum Classification Algorithms

Farid Ablayev, Marat Ablayev, Joshua Zhexue Huang, Kamil Khadiev, Nailya Salikhova, and Dingming Wu*

Abstract: This is a review of quantum methods for machine learning problems that consists of two parts. The first part, "quantum tools", presented some of the fundamentals and introduced several quantum tools based on known quantum search algorithms. This second part of the review presents several classification problems in machine learning that can be accelerated with quantum subroutines. We have chosen supervised learning tasks as typical classification problems to illustrate the use of quantum methods for classification.

Key words: quantum classification; binary classification; nearest neighbor algorithm

1 Introduction

Classification is one of the basic problems for Machine Learning (ML). There are many applications for classification tasks, so the problem has been intensively researched. In this part of the review, we present variants of Nearest Neighbour (NN) algorithms for classification problems and two algorithms for binary classification which is a specific case of the classification problem.

1.1 Algorithms for the binary classification

This review presents two well-known methods for binary classification: Support Vector Machine (SVM)[1] and perceptron[2, 3]. The main idea of the two methods is constructing a hyperplane in the feature space to separate training vectors into two classes. However, the two methods have different ways of choosing a hyperplane.

1.1.1 SVM

The main computational step of the SVM algorithm is the solving of a quadratic programming problem that is NP-hard. The only solution is discretization and a brute-force algorithm. The quantum algorithm[4] uses the quantum maximum search[5], which is based on the quantum Grover's search algorithm[6]. It achieves a quadratic speed-up compared to the known classical algorithm.

1.1.2 Perceptron

The perceptron method is based on the idea of a random sampling of hyperplanes and checking whether a candidate hyperplane classifies training data well. The quantum algorithm for candidate hyperplane checking also uses the quantum Grover's search algorithm and again provides a quadratic speed-up compared to the classical algorithm[7, 8].

1.2 Nearest neighbour algorithms

The k-Nearest Neighbor (k-NN) algorithm relates an object to the class of the majority of its k-nearest neighbors in the multidimensional feature space. In the learning process, the distances between the unclassified objects and those that have been previously classified are calculated. The main disadvantage of this classical algorithm is its high computational complexity, which increases quadratically with the number of objects.

Obviously, the k-NN algorithm's solution to the classification problem has an inherent parallelism. This fact is exploited in quantum approaches to k-NN

• Joshua Zhexue Huang and Dingming Wu are with the College of Computer Science & Software Engineering, Shenzhen University, Shenzhen 518000, China. E-mail: zx.huang@szu.edu.cn; dingming@szu.edu.cn.
• Farid Ablayev, Marat Ablayev, Kamil Khadiev, and Nailya Salikhova are with the Kazan Federal University, Kazan 42008, Russia. E-mail: fablayev@gmail.com; mablayev@gmail.com; kamilhadi@gmail.com; nailyasalikhova66@gmail.com.
* To whom correspondence should be addressed.
Manuscript received: 2019-09-10; accepted: 2019-09-25

classification. In the quantum case, thanks to the phenomenon of quantum superposition, we can calculate the distances between the unclassified object and all other objects in the quantum parallel mode, and encode this information in amplitudes. A partial measurement of the corresponding qubits of the quantum register with a certain probability gives information about the most appropriate class label for the unclassified object.

Section 4 presents two variants of quantum versions of k-NN algorithms. The Hamming distance is used as a metric for vector proximity. Both algorithms can be split into two stages: the preliminary stage and the main stage. The preliminary stage of each algorithm (the procedure presented in Section 4.1) is an algorithm in the quantum branching program model that is used for reading data. The main stage of each algorithm is a quantum circuit that does not read input data. The first k-NN quantum algorithm[9] is an analog of the classical weighted NN algorithm, where k is a number of all training vectors. The second k-NN quantum algorithm[10] is an analog of the classical NN with one parameter: the threshold value t. This threshold is used for selecting nearest neighbors, based on a Hamming distance from the test vector of at most t.

The third quantum algorithm presented in Section 5 is another interpretation of the NN algorithm. This algorithm is presented in a quantum query model. It works with sparse training vectors and can show up to quadratic speed-up comparing to the classical algorithm[11]. The main idea of the classical approach is to choose the nearest neighbor and assign its class to the test vector (i.e., it is the k-NN algorithm for $k = 1$). The quantum method can speed-up the nearest neighbor vector search using the quantum algorithm for maximum search, which is based on Grover's search algorithm.

2 Preliminaries

The classification problem is the task of assigning a new object to one of a set of predefined classes $\{0, 1, \ldots, D-1\}$. Each object is represented as a vector in the $\mathbb{R}^n$ space, where each element of the vector is a description of one of the object's features. A training set of input vectors $x^1, \ldots x^m \in \mathbb{R}^n$ is required, with the corresponding output values $y^1, \ldots y^m \in \{0, \ldots D-1\}$.

The NN method is a well-known approach to classification and clustering problems. We consider the classification problem[12, 13] in this review. The algorithm is a metric algorithm, meaning that a metric is defined in the feature space. Its essence lies in the idea that the new object belongs to the same class as its nearest neighbor from the training set. The k-NN method generalizes the NN method by stipulating that the new object belongs to the class of the majority of its k-nearest neighbors. In classification problems with an even number of classes, k is an odd number, so that the same number of nearest neighbors will not have different classes. Problems with an odd number of classes, give rise to an ambiguity problem that can be solved by applying the weighted nearest neighbor method, in which the value of a weight function w_i is assigned to the i-th nearest neighbor of the new object.

The weight function is one of the parameters in the weighted nearest neighbor method that is chosen by the researcher. It assesses the degree of importance of the i-th neighbor for classifying a new object or test object. This function is non-negative and does not increase in i. It may depend on the distance (metric value) from the new object to its i-th nearest neighbor $\omega_i = \omega(d(x, x^i))$, where x is a new object; or from the number of the nearest neighbor $\omega_i = \omega(i)$. For example, as a weight function, depending on the number of the nearest neighbor of a new object, a geometric progression $\omega(i) = (\delta_{i \leqslant k}) \cdot q^i$, $0.5 \leqslant q \leqslant 1$ can be taken, where $\delta_{\text{condition}}$ is 1 if the condition is true and 0 otherwise. The definition of ω means that the element with the higher index has a larger weight, but elements with indexes greater than k do not have any weight.

We thereby obtain a variant of the method of k-exponentially weighted nearest neighbors, in which the new object belongs to the class that has the greatest total weight among its k-nearest neighbors. If $\omega(i) = 1$ and $i < k$, then the algorithm corresponds to the method of k-NN.

In general, the weighted nearest neighbor algorithm can be presented as follows:

$$y = \underset{y \in Y}{\operatorname{argmax}} \sum_{i \in \{1,\ldots,k\}, \text{ and } y^i = y} \omega_i \qquad (1)$$

where $Y = \{0, 1, \ldots, D-1\}$ and y is the class label for the new object. The number of neighbors k and the metrics are parameters of the model, and are chosen by the researcher.

3 Quantum Algorithms for Binary Classification

We start with binary classification because it is the

simplest version of the problem, but the algorithms can be modified to deal with more classes. Here we present the problem formally.

Binary classification problem. Given a sequence of input vectors $x^1, \ldots, x^m \in \mathbb{R}^n$ and a sequence of corresponding output values $y^1, \ldots, y^m \in \{-1, 1\}$, find a function $F : \mathbb{R}^n \to \{-1, 1\}$ that classifies an input data item.

3.1 SVM

The SVM is a well-known algorithm for the classification problem[1]. For function F, we can write the following:

$$F(x) = \text{sign}\left(\sum_{i=1}^{n} x_i w_i + w_0\right) = \text{sign}((x \cdot w) + w_0) \tag{2}$$

where $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$, the coefficients $w = (w_1, \ldots, w_n) \in \mathbb{R}^n$ and $w_0 \in \mathbb{R}$ are parameters of the algorithm, $(x \cdot w)$ is the inner product of the vectors, and the hyperplane $(x \cdot w) + w_0$ separates two classes.

As a plane, we choose the farthest one of the nearest vectors. The closest vectors to the plane are called "support vectors" and should be such that $(x^i \cdot w) + w_0 = y^i$ or, equivalently, $y^i((x^i \cdot w) + w_0) = 1$. For finding the parameters of the plane, we should maximize the distance between different classes with $2/||w||$ or, equivalently, minimize $0.5||w||^2$. The minimization problem can be solved with the following conditions:

$$y^i((x^i \cdot w) + w_0) \geqslant 1,\ i \in \{1, \ldots, m\} \tag{3}$$

Based on the Kuhn–Tucker theorem, we can replace the problem by the dual formulation of the Lagrange function:

$$\mathcal{L}(w, w_0, \lambda) = 0.5(w \cdot w) - \sum_{i=1}^{m} \lambda_i (y^i((w \cdot x^i) - w_0) - 1) \to \min_{w \in \mathbb{R}^n, w_0 \in \mathbb{R}} \max_{\lambda \in \mathbb{R}^m} \tag{4}$$

with the following conditions for $i \in \{1, \ldots, m\}$:

- $\lambda_i \geqslant 0$;
- $\lambda_i = 0$, or $(x^i \cdot w) - w_0 = y_i$.

At the same time, we have the following conditions:

$$\begin{cases} \dfrac{\partial \mathcal{L}}{\partial w} = w - \displaystyle\sum_{i=1}^{m} \lambda_i y^i x^i = 0; \\ \dfrac{\partial \mathcal{L}}{\partial w_0} = -\displaystyle\sum_{i=1}^{m} \lambda_i y^i = 0 \end{cases} \tag{5}$$

Therefore, we obtain the next relation between λ and w:

$$\begin{cases} w = \displaystyle\sum_{i=1}^{m} \lambda_i y^i x^i; \\ \displaystyle\sum_{i=1}^{m} \lambda_i y^i = 0 \end{cases} \tag{6}$$

Finally, we obtain the following optimization problem on the λ variables:

$$\mathcal{L}(\lambda) = \sum_{i=1}^{m} \lambda_i + 0.5 \sum_{i=1}^{m} \sum_{j=1}^{m} \lambda_i \lambda_j y^i y^j (x^i \cdot x^j) \to \max_{\lambda} \tag{7}$$

with the following conditions for $i \in \{1, \ldots, m\}$:

- $\lambda_i \geqslant 0$;
- $\sum_{i=1}^{m} \lambda_i y^i = 0$.

In a case of a non-linear separation between classes, we can add errors ξ_i; in that case $y^i((w \cdot x^i) + w_0) \geqslant 1 - \xi_i$. The minimization function is replaced by $0.5||w||^2 + C \sum_{i=1}^{m} \xi_i$, where C is a constant that is a parameter of the algorithm. The Lagrange function in Eq. (4) is replaced by

$$\mathcal{L}(w, w_0, \sigma, \lambda, \beta) = 0.5||w||^2 + C \sum_{i=1}^{m} \xi_i - \sum_{i=1}^{m} \lambda_i \left\{\lambda_i \left[(w \cdot x^i) - w_0\right] - 1 + \xi_i\right\} - \sum_{i=1}^{m} \beta_i \xi_i \tag{8}$$

with λ_i and $\beta_i \geqslant 0$. The relation between λ, β and w, ξ can be obtained using an equation similar to Eq. (6). At the same time, we obtain the additional equality $C - \beta_i - \lambda_i = 0$. Because $\beta_i \geqslant 0$, we get $\lambda_i \leqslant C$. The final minimization problem is

$$\mathcal{L}(\lambda) = \sum_{i=1}^{m} \lambda_i + 0.5 \sum_{i=1}^{m} \sum_{j=1}^{m} \lambda_i \lambda_j y^i y^j (x^i \cdot x^j) \to \max_{\lambda} \tag{9}$$

with the following conditions for $i \in \{1, \ldots, m\}$:

- $C \geqslant \lambda_i \geqslant 0$;
- $\sum_{i=1}^{m} \lambda_i y^i = 0$.

If the input data is not linearly separable, then we can replace the inner product $(x^i \cdot x^j)$ by another function $k(x^i, x^j)$, which is called the kernel of the SVM algorithm. Typically, researchers choose the following functions as possible kernels: $(x^i \cdot x^j)^d$ and $(x^i \cdot x^j + 1)^d$ for some constant d; or $e^{-\gamma||x^i - x^j||^2}$ for some constant $\gamma > 0$.

In the general case, $\mathcal{L}(\lambda)$ can have several local

minimums, and gradient-based methods are unhelpful. In that case, the problem is NP-hard. We can apply discretization and then use brute force or analog algorithms. We can also apply the Grover's search to this problem and obtain a quadratic speed-up. A description of the algorithm follows.

Supposing that $\varepsilon > 0$ is a given precision for our solution, then we can split the segment $[0; C]$ to points with interval ε; these points will be possible solutions. There are $d = 1 + \frac{C}{\varepsilon}$ points for one λ_i, so the j-th point is $\lambda_i = \varepsilon \cdot j$, for $j \in \{0, \ldots, m\}$. Enumerating all possible solutions $(\lambda_1, \ldots, \lambda_m)$ with precision ε, the solution number $q = \sum_{i=1}^{m} j_i \cdot (d+1)^{i-1}$ is $(\lambda_1^q, \ldots, \lambda_m^q) = (\varepsilon \cdot j_1, \ldots, \varepsilon \cdot j_m)$ for $j_i \in \{0, \ldots, d\}$ and $i \in \{1, \ldots, m\}$. We can obtain q by $(j_1, \ldots, j_m)$ and vice versa.

With the function $f : \{0, \ldots, (d+1)^m - 1\} \to \mathbb{R}$ as $f(q) = \mathcal{L}(\lambda_1^q, \ldots, \lambda_m^q)$, the problem is a maximum search for the function f. The quantum version of the algorithm is presented as Algorithm 1 (QSVM represents Quantum Supporting Vector Machine).

The complexity of the quantum algorithm is presented in Property 1 and can be compared with the complexities given in the subsequent properties.

Property 1. The expected quantum query complexity of the quantum version of the learning phase for the SVM algorithm is $\text{height(QSVM)} = O\left(\left(\frac{C}{\varepsilon}\right)^{m/2} m^2 + m^2 n\right)$.

Proof. $O(m^2 n)$ is required for precomputing all of the inner products. The maximum search algorithm then computes the function f. The complexity of computing f is $O(m^2)$ and the size of the search space is $\left(\frac{C}{\varepsilon}\right)^m$. Hence, the complexity of the maximum search is $O\left(\left(\frac{C}{\varepsilon}\right)^{m/2} m^2\right)$ and the total complexity is $O\left(\left(\frac{C}{\varepsilon}\right)^{m/2} m^2 + m^2 n\right)$.

Algorithm 1 **QSVM $(x^1, \ldots, x^m)$. Quantum version of the SVM learning phase.**

```
for i ∈ {1, ..., m} do                ▷ Precomputing all inner products (x^i · x^j)
    for j ∈ {1, ..., m} do
        ip_{i,j} ← (x^i · x^j)
    end for
end for
q ← Grover_max(0, (d + 1)^m − 1, f)
λ ← λ_1^q, ..., λ_m^q
w ← Σ_{i=1}^{m} λ_i y^i x^i
return w
```

For comparison, the classical algorithm has the following complexity.

Property 2. The query complexity of the learning phase for the SVM algorithm is $\text{height(SVM)} = O\left(\left(\frac{C}{\varepsilon}\right)^m m^2 + m^2 n\right)$.

Note that the different approach to the solution given in Ref. [14] achieves an exponential speed-up.

3.2 Perceptron

The perceptron is a simple Artificial Neural Network (ANN). It can also be considered as a building block for ANNs[2] and as a modification of the SVM algorithm[3]. Wiebe et al.[7,8] suggested a quantum algorithm for training a classical perceptron that shows a quadratic speed-up. The algorithm is based on Grover's search algorithm[6].

In the classical model, the function F can be written as follows:

$$F(x) = \text{sign}\left(\sum_{i=1}^{n} x_i w_i + w_0\right) = \text{sign}(x \cdot w + w_0) \tag{10}$$

where $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ is the input, $w = (w_1, \ldots, w_n) \in \mathbb{R}^n$ and $w_0 \in \mathbb{R}$ are parameters of the algorithm; $x \cdot w$ is the inner product of the vectors, the hyperplane $x \cdot w + w_0$ separates two classes, and w and w_0 should be chosen such that $y^i(w \cdot x^i + w_0) > 0$. There is an assumption that all training vectors are separated by a margin of γ in the feature space.

The idea of the classical algorithm is to invoke random sampling of (w, w_0) and check whether the hydroplane classifies the training set correctly. The classical algorithm requires $O(mn)$ to search for the existence of a pair (x^i, y^i) that is misclassified; that is, $y^i(w \cdot x^i + w_0) < 0$. $O\left(\log \frac{1}{\varepsilon\gamma^2}\right)$ samples need to be checked to get the correct one with error probability $\varepsilon\gamma^2$ for some constant $\varepsilon > 0$.

Property 3. The query complexity of sampling algorithm A for perceptron is $\text{height(A)} = O\left(nm \log \frac{1}{\varepsilon\gamma^2}\right)$, where $\varepsilon > 0$ is a constant and the training vectors are separated by a margin of γ in the feature space.

Quantum algorithm for perceptron training. The quantum algorithm operates with a quantum register $|\psi\rangle$ of $\lceil \log_2 m \rceil$ qubits. As a function for oracle, we use $f_w(i) = \mathcal{F}_w(x^i, y^i)$, where $\mathcal{F}_w(x^i, y^i)$ is a Boolean-value function that is 1 if and only if the vector x^i is misclassified. We use the amplitude

amplification version of the Grover's search algorithm to find misclassified vectors. The query complexity of the procedure is $O(\sqrt{mn})$. As in the classical case, we invoke random sampling of (w, w_0) for $K = O\Big(\log \frac{1}{\varepsilon\gamma^2}\Big)$ times. The algorithm is presented as Algorithm 2.

As was proven in Ref. [11], if $K = \log_{3/4} \varepsilon$ for some $\varepsilon > 0$, then the algorithm returns the result with error probability $\varepsilon\gamma^2$. The complexity of the algorithm is presented in the following property.

Property 4. The query complexity of Algorithm 2 is height(QPerceptron_Trnning) $= O\Big(n\sqrt{m}\log\frac{1}{\varepsilon\gamma^2}\Big)$, where $\varepsilon > 0$ is a constant and the training vectors are separated by a margin of γ in the feature space.

4 Quantum Nearest Neighbour Algorithms

We consider the classification problem for D classes, assuming that the input vectors are binary or at least represented in binary form. Here we present the problem formally.

Problem. Given the sequence of input vectors $x^1, \ldots, x^m \in \{0,1\}^n$ and the sequence of corresponding output values $y^1, \ldots, y^m \in \{0, \ldots, D-1\}$, find a function $F : \{0,1\}^n \to \{0, \ldots, D-1\}$ that classifies an input data item.

We will call $x^1, \ldots, x^m, y^1, \ldots, y^m$ the training data and $x^1, \ldots, x^m$ the training vectors. The argument of F is a test vector.

4.1 Training set superposition construction subroutine

A utility procedure will be used in the quantum algorithms of this section. The procedure is an algorithm in the quantum branching program model, and it is required to store a superposition of training vectors in a single quantum state of n qubits; that is, it needs to get the following quantum state:

$$|T\rangle = \frac{1}{\sqrt{m}}\sum_{j=1}^{m}|x^j\rangle \tag{11}$$

Trugenberger[15] introduced an algorithm for generating this state unitarily from a simple initial state of n qubits.

Property 5. If n is an input vector size and m is the number of input vectors in a training set, then there is an algorithm A for constructing the quantum state $|T\rangle$. The algorithm has the following complexity properties:

- size(A) $= 2n + 2$;
- height(A) $= m$;
- time(A) $= O(mn)$.

Let us construct the algorithm A as the proof of the property.

We use three registers. The first is a register $|\varphi\rangle$ of n qubits for storing input vectors x^j; the second register is a utility register $|u\rangle = |u_1\rangle\,|u_2\rangle$ of two qubits; and the third is a register $|v\rangle$ of n qubits that is holding memory. The full initial state is

$$|\psi_0^1\rangle = |\varphi\rangle\,|u\rangle\,|v\rangle = |0, \ldots, 0; 01; 0, \ldots, 0\rangle \tag{12}$$

This state is split into two terms: one corresponding to the already stored input vectors and another ready to process a new input vector from the training set. These two parts are distinguished by the state of the second utility qubit $|u_2\rangle$. The $|0\rangle$ state is for the stored input vectors, and $|1\rangle$ is readiness for processing.

Before processing the vector x^j, the algorithm reads it and stores it to the $|\varphi\rangle$ register. This can be done using the *CNOT* operator. The controlling bit is an input bit x_s^j and the target bit is the s-th qubit of $|\varphi\rangle$, where $s \in \{1, \ldots, n\}$. If $|\varphi\rangle = |0, \ldots, 0\rangle$ before the procedure, then after the procedure $|\varphi\rangle = |x_1^j, \ldots, x_n^j\rangle$. If $|\varphi\rangle = |x_1^j, \ldots, x_n^j\rangle$ before the procedure, then after the procedure $|\varphi\rangle = |0, \ldots, 0\rangle$. Therefore, we can use the same procedure for erasing the $|\varphi\rangle$ register. That is why we consider an input with the duplication of each vector x^j; that is, the input is $x^1, x^1, x^2, x^2, \ldots, x^m, x^m$. For the next part of the procedure, we assume that $|\varphi\rangle = |x_1^j, \ldots, x_n^j\rangle$.

For each input vector x^j to be stored, the operations described below are performed:

$$|\psi_1^j\rangle = \prod_{s=1}^{n} 2CNOT(x_s^j, u_2, v_s)|\psi_0^j\rangle \tag{13}$$

Algorithm 2 **QPerceptron_Trnning $(x, (x^1, \ldots, x^m))$. Quantum version of the preception training procedure.**

found ← False
for $i \in \{1, \ldots, K\}$ **do**
  $(w, w_0) \leftarrow$ Random($\mathcal{N}(0,1)$)
  $j \leftarrow$ Grover$(1, m, \mathcal{F}_{(w,w_0)})$
  if $j = -1$ **then**
    found ← True
    break; ▷ Stopping the for-cycle
  end if
end for
return (w, w_0)

where x_s^j and u_2 are the control qubits, and v_s is the target qubit.

This operation simply copies the input vector x^j into the memory register of the processing term, identified by $|u_2\rangle = |1\rangle$.

The next operation works as follows. If the contents of the input vector and the memory register are identical, then all of the qubits of the memory register are set to $|1\rangle$ (this will be exactly the case only for the processing term):

$$|\psi_2^j\rangle = \prod_{s=1}^{n} NOT(v_s)CNOT(x_s^j, v_s)|\psi_1^j\rangle \qquad (14)$$

where in the *CNOT* operator, x_s^j is the control qubit and v_s is the target qubit.

The next operation changes the first utility qubit u_1 of the processing term to a $|1\rangle$, while keeping it unchanged for the stored input vectors term:

$$|\psi_3^j\rangle = nCNOT(v_1, \ldots, v_n, u_1)|\psi_2^j\rangle \qquad (15)$$

where $v_1, \ldots, v_n$ are control qubits and u_1 is the target qubit.

This is followed by the controlled normalization operator:

$$\begin{cases} CS^j = |0\rangle\langle 0| \otimes 1 + |1\rangle\langle 1| \otimes S^j; \\ S^j = \begin{pmatrix} \sqrt{\dfrac{j-1}{j}} & \dfrac{1}{\sqrt{j}} \\ \dfrac{-1}{\sqrt{j}} & \sqrt{\dfrac{j-1}{j}} \end{pmatrix} \end{cases} \qquad (16)$$

for $j = 1, \ldots, m$

$$|\psi_4^j\rangle = CS^{m+1-j}(u_1, u_2)|\psi_3^j\rangle \qquad (17)$$

This operation separates out the new input vector to be stored with the already correct normalization factor.

The utility qubit u_1 is then restored to its original value:

$$|\psi_5^j\rangle = nCNOT(v_1, \ldots, v_n, u_1)|\psi_4^j\rangle \qquad (18)$$

where $v_1, \ldots, v_n$ are control qubits and u_1 is the target qubit.

The next operation restores the memory register v to its original state:

$$|\psi_6^j\rangle = \prod_{s=n}^{1} CNOT(x_s^j, v_s)NOT(v_s)|\psi_5^j\rangle \qquad (19)$$

where in the *CNOT* operator, x_s^j is the control qubit and v_s is the target qubit.

When these operations are complete,

$$|\psi_6^j\rangle = \frac{1}{\sqrt{m}} \sum_{k=1}^{j} |x^j; 00; x^k\rangle + \sqrt{\frac{m-j}{m}}|x^j; 01; x^j\rangle \qquad (20)$$

The third register v of the processing term and the second term in the equation above are then restored to the initial value $|0_1, \ldots, 0_n\rangle$:

$$|\psi_7^j\rangle = \prod_{s=n}^{1} 2CNOT(x_s^j, u_2, v_s)|\psi_6^j\rangle \qquad (21)$$

where x_s^j and u_2 are control qubits, and v_s is the target qubit.

At this point, a new input vector can be loaded into register $|\varphi\rangle$ and the sequence of operations reiterated. At the end of the whole process, the v-register is exactly in the state $|T\rangle$.

The time complexity of the algorithm is $O(mn)$.

The algorithm is presented as Algorithm 3, where X is a training set, m is the length of the training set, and n is the length of each input vector in the training set. The algorithm constructs a superposition of input vectors of the training set X.

In the next two subsections, we describe two quantum k-NN algorithms. The first is from Ref. [9] and the second is from Ref. [10]. Both algorithms can be split into two stages: the preliminary stage and the main stage.

The similarity of these two algorithms lies in the preliminary stage that prepares a superposition of the training data using Algorithm 3. Their differences are as follows:

- They have different mechanisms for computing the distance between vectors;
- The first algorithm changes the superposition of the training data in the main stage, while the second algorithm does not change it until the measurement.

Since the preliminary stage of both algorithms has the time complexity $O(mn)$, these two algorithms have the

Algorithm 3 Construct_Superposition(X, m, n). Quantum algorithm for constructing superposition of input vectors of the training set.

$|\psi^0\rangle = |\varphi; u; v\rangle =\leftarrow |0, \ldots, 0; 01; 0, \ldots, 0\rangle$ ▷ The initial value
for $j \in \{1, \ldots, m\}$ **do**
  load x^j into the register $|\varphi\rangle$ of $|\psi^{j-1}\rangle$ ▷ result state is denoted as $|\psi^j\rangle$
  $|\psi^j\rangle \leftarrow \prod_{s=1}^{n} 2CNOT(x_s^j, u_2, v_s)|\psi^j\rangle$
  $|\psi^j\rangle \leftarrow \prod_{s=1}^{n} NOT(v_s)CNOT(x_s^j, v_s)|\psi^j\rangle$
  $|\psi^j\rangle \leftarrow nCNOT(v_1, \ldots, v_n, u_1)|\psi^j\rangle$
  $|\psi^j\rangle \leftarrow CS^{m+1-j}(u_1, u_2)|\psi^j\rangle$
  $|\psi^j\rangle \leftarrow nCNOT(v_1, \ldots, v_n, u_1)|\psi^j\rangle$
  $|\psi^j\rangle \leftarrow \prod_{s=n}^{1} CNOT(x_s^j, v_s)NOT(v_s)|\psi^j\rangle$
  $|\psi^j\rangle \leftarrow \prod_{s=n}^{1} 2CNOT(x_s^j, u_2, v_s)|\psi^j\rangle$
  erase $|\varphi\rangle$ using x^j
end for
return $|v\rangle$ ▷ superposition in the register v

same time complexity as their classic counterparts. If a more efficient way to construct the superposition in $O(n)$, or to receive the superposition from a quantum memory device, were to be found, the performance of these quantum algorithms would be independent of the number of training vectors. It is impossible to achieve such an effect with the classical algorithm.

Additionally, if a method of training vectors storage uses a smaller number of qubits, without affecting the proportions between the features of the vectors, then it would be possible to reduce the memory size for these algorithms.

4.2 Schuld-Sinayskiy-Petruccione (SSP) algorithm

The algorithm, presented in Ref. [9], is a quantum version of the classical weighted nearest neighbor algorithm. The idea of the quantum version of the algorithm is as follows. Firstly, we prepare a superposition of all vectors from the training set as one quantum state, using the procedure from Section 4.1. Second, we compute the Hamming distance between each training vector and the test vector, and store this into the amplitude of each training vector in the superposition. Having done that, if we measure the class qubit, we will get the appropriate class with high probability. If we repeat the algorithm enough times, then we can get the probability distribution, which will carry information about the average closeness of members of each class to the test vector ("$k \leftarrow$ all").

Only the preliminary stage of the algorithm (the procedure in Section 4.1) reads input variables; the main stage is a quantum circuit that does not read input data.

Property 6. If n is the size of an input vector, m is the number of input vectors in a training set, and D is a number of classes, then the algorithm SSP–kNN for classifying a test vector has the following complexity properties:

- size(SSP-kNN) $= 2n + \log D + 1$;
- time(SSP-kNN preliminary stage) $= O(mn)$;
- time(SSP-kNN main stage) $= O(n)$.

The SSP algorithm can be described as follows.

The first step of the algorithm is preparing a superposition of all of the training vectors into one quantum state using the construction of a training set superposition algorithm from Section 4.1.

$$|T\rangle = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} |x_1^j, \ldots, x_n^j, y^j\rangle \tag{22}$$

The algorithm for constructing the superposition of vectors of the training set has $O(mn)$ time complexity. The test vector is represented as $|x\rangle = |x_1, \ldots, x_n\rangle$.

The second step is to prepare the following initial state:

$$|\psi_0\rangle = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} |x_1, \ldots, x_n; x_1^j, \ldots, x_n^j, y^j; 0\rangle \tag{23}$$

This state is made up of three registers: The first contains the test vector (n qubits), the second contains the superposition of the training vectors ($n + \lceil \log D \rceil$ qubits), and the third contains a utility qubit. The part of the second register that stores y^j and contains $\lceil \log D \rceil$ qubits will be called the "class register". Initially, we assign $|0\rangle$ to the last utility qubit.

The third step of the algorithm is to apply the Hadamard gate to the utility qubit:

$$|\psi_1\rangle = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} |x_1, \ldots, x_n; x_1^j, \ldots, x_n^j, y^j\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \tag{24}$$

The fourth step is to prepare a state for obtaining the Hamming distance. We apply the *CNOT* gate to $|x_1, \ldots, x_n; x_1^j, \ldots, x_n^j\rangle$ with $|x_1, \ldots, x_n\rangle$ as the control register and $|x_1^j, \ldots, x_n^j\rangle$ as a target register, and then apply the *NOT* gate to reverse the states in the second register:

$$|\psi_2\rangle = \prod_{s=1}^{n} NOT(x_s^j) CNOT(x_s, x_s^j) \, |\psi_1\rangle = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} |x_1, \ldots, x_n; d_1^j, \ldots, d_n^j, y^j\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \tag{25}$$

where

$$d_s^j = \begin{cases} 1, \text{ if } |x_s^j\rangle = |x_s\rangle; \\ 0, \text{ otherwise.} \end{cases}$$

The fifth step is to apply the unitary operator U, where

$$U = \mathrm{e}^{\mathrm{i}\frac{\pi}{2n}\eta}, \quad \eta = I \otimes D_{\mathrm{Ham}}^j \otimes I \otimes \sigma_z,$$

$$D_{\mathrm{Ham}}^j = \sum_{s=1}^{n} \left(\frac{\sigma_z + I}{2} \right),$$

in which σ_z is a Pauli-Z matrix. The operator η assigns the number of zeros in the second register as a phase in the case of a $|0\rangle$ value of the utility qubit. In the case of a $|1\rangle$ value of the utility qubit, the operator assigns the same value, but as a negative. Note that the number of zeros is the number of unequal bits for $|x\rangle$ and $|x^j\rangle$; that is, the Hamming distance between these vectors.

Denoting this as $d_{\text{Ham}}(x, x^j)$, the state after applying η is

$$|\psi_3\rangle = U\,|\psi_2\rangle = \frac{1}{\sqrt{2m}}\sum_{j=1}^{m} (\mathrm{e}^{\mathrm{i}\frac{\pi}{2n}d_{\text{Ham}}(x,x^j)}|x_1,\ldots,x_n;d_1^j,\ldots,d_n^j,y^j;0\rangle + \mathrm{e}^{-\mathrm{i}\frac{\pi}{2n}d_{\text{Ham}}(x,x^j)}|x_1,\ldots,x_n;d_1^j,\ldots,d_n^j,y^j;1\rangle) \quad (26)$$

The sixth step is to apply a Hadamard gate on the utility qubit, and write the phase information of the j-th state into amplitudes:

$$\begin{cases} U_2 = I\otimes I\otimes I\otimes H; \\ |\psi_4\rangle = U_2\,|\psi_3\rangle = \dfrac{1}{\sqrt{m}}\displaystyle\sum_{j=1}^{m} \\ \Big(\cos\frac{\pi}{2n}d_{\text{Ham}}(x,x^j)|x_1,\ldots,x_n;d_1^j,\ldots,d_n^j,y^j;0\rangle + \\ \sin\frac{\pi}{2n}d_{\text{Ham}}(x,x^j)|x_1,\ldots,x_n;d_1^j,\ldots,d_n^j,y^j;1\Big) \end{cases} \quad (27)$$

The seventh step is to measure the utility qubit. If the test vector is close to the training vector, then the probability of a $|0\rangle$ result is high; otherwise, the probability of a $|1\rangle$ result is high. The probability of getting $|0\rangle$ is

$$Pr_0 = \frac{1}{\sqrt{m}}\sum_{j=1}^{m}\cos^2\Big[\frac{\pi}{2n}d_{\text{Ham}}(x,x^j)\Big] \quad (28)$$

In other words, if there are many training vectors for which the Hamming distance between them and the test vector is small, then Pr_0 is high.

If we obtain a $|0\rangle$ result, then the next step is to measure the "class register". We have a higher probability of getting a class $c \in \{0,1,\ldots,D-1\}$ for classes that have training vectors with smaller Hamming distances from the test vector. After obtaining a $|0\rangle$ result from the utility qubit measurement, we have the following state:

$$|\psi_5\rangle = \frac{1}{\sqrt{m\cdot Pr_0}}\sum_{y=0}^{D-1}|y\rangle\otimes\sum_{l:y^l=y}\cos\frac{\pi}{2n}d_{\text{Ham}}(x,x^l)|x_1,\ldots,x_n;d_1^l,\ldots,d_n^l,y^l;0\rangle \quad (29)$$

The probability of obtaining c-result from the "class register" measurement is

$$Pr(c) = \frac{1}{m\cdot Pr_0}\sum_{l:y^l=c}\cos^2\Big[\frac{\pi}{2n}d_{\text{Ham}}(x,x^l)\Big] \quad (30)$$

The class c has a higher probability where more training vectors of this class have a small Hamming distance from the test vector.

Since for small angles θ $\left(\theta < \dfrac{\pi}{12}\right)$, $\sin\theta \approx \theta$, and $\cos\theta \approx 1-\theta^2$, the following statement is true.

(1) If for b vectors among m vectors of the training set

$$d_{\text{Ham}} < \frac{n}{6},$$

then the following double inequality for the probability Pr_0 is true:

$$\frac{b}{m}\left[1-\left(\frac{\pi}{12}\right)^2\right] < Pr_0 \leqslant 1.$$

(2) If b_c vectors among above b vectors belong to c class, then the following double inequality for probability $Pr(c)$, conditioned on the probability Pr_0, is true:

$$\frac{b_c}{b} < Pr(c) \leqslant 1.$$

For computing the time complexity of the algorithm, there are n *NOT* gates and n *CNOT* gates in the fourth step, while the fifth step requires $2n$ gates for the unitary operator U implementation due to Ref. [16], and there are two Hadamard gates in Steps 3 and 6. The total time complexity of the main stage of the algorithm is therefore $4n+2$.

The algorithm is presented as Algorithm 4 and is labelled as SSP-kNN(X,m,n,x), where X is a training set, m is the length of the training set, n is the length of each vector in the training set, and x is the test vector. If we obtain a 1-result from the utility qubit measurement, then we are unable to obtain a class, and the algorithm

Algorithm 4 SPP-kNN(X, m, n, x). Schuld-Sinayskiy-Petruccione algorithm.

$|T\rangle \leftarrow$ Construct_Superposition(X,m,n) ▷ Constructing the superposition of the training set X: $\frac{1}{\sqrt{m}}\sum_{j=1}^{m}|x_1^j,\ldots,x_n^j,y^j\rangle$
$|x\rangle \leftarrow |x_1,\ldots,x_n\rangle$ ▷ test vector
$|u\rangle \leftarrow |0\rangle$ ▷ utility qubit
$|\psi\rangle \leftarrow |x\rangle\,|T\rangle\,|u\rangle$ ▷ initial state
$|\psi\rangle \leftarrow |x\rangle\,|T\rangle\otimes H\,|u\rangle$ ▷ H gate to the utility qubit
$|\psi\rangle \leftarrow \prod_{s=1}^{n} X(x_s)CNOT(x_s,x_s^j)|\psi\rangle$
$|\psi\rangle \leftarrow U|\psi\rangle$ ▷ applying operator U for calculating the Hamming distance
$|\psi\rangle \leftarrow |x\rangle\,|T\rangle\otimes H|u\rangle$
$ut \leftarrow$ Measurement$(|u\rangle)$ ▷ measuring the utility qubit
if ut is $|0\rangle$ **then**
  $c \leftarrow$ Measurement$(|y\rangle)$ ▷ measure register $|y\rangle$
end if
if ut is $|1\rangle$ **then**
  $c \leftarrow -1$
end if
return c

returns -1 to represent "unknown".

This algorithm is run enough times as necessary to obtain a precise picture from the measurement results.

4.3 Ruan-Xue-Liu-Tan-Li (RXLTL) algorithm

In this algorithm, which was presented in Ref. [10], we are provided with a Hamming distance threshold value t. The RXLTL–NN algorithm finds all of the training vectors for which the Hamming distance from the test vector is at most t, then it assigns the most frequent class among the chosen set of training vectors to the test vector.

Property 7. If n is the size of a vector, D is the number of classes, m is the number of training vectors, and t is the threshold value for the distance between the test vector and the vectors of the training set ($t \leqslant n$), then the algorithm RXLTL–NN for classifying the test vector has the following complexity properties:

- size(RXLTL-NN) $= 2n + \lceil \log_2 n \rceil + \log D + 1$;
- time(RXLTL-NN preliminary step) $= O(mn)$;
- time(RXLTL-NN main steps) $= O(n^3)$.

The proof of the property is attained by describing RXLTL algorithm below.

The first step of the algorithm is to prepare a superposition of all training vectors into one quantum state using the construction of the training set superposition algorithm outlined in Section 4.1.

$$|T\rangle = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} |x_1^j, \ldots, x_n^j, y^j\rangle \tag{31}$$

The time complexity of this step is $O(mn)$. The training set superposition does not change in the main stage until the measurement step. The test vector is represented as $|x\rangle = |x_1, \ldots, x_n\rangle$.

The second step is to prepare the following initial state:

$$|\psi_0\rangle = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} |x_1, \ldots, x_n; x_1^j, \ldots, x_n^j, y^j\rangle \tag{32}$$

This state is made up of two registers. The first contains the test vector (n qubits) and the second is the superposition of the training vectors ($n + \lceil \log D \rceil$ qubits). The part of the second register that stores y^j and contains $\lceil \log D \rceil$ qubits will again be called the "class register".

The third step is to prepare the state for computing the Hamming distance. We apply a *CNOT* operator to $|x_1, \ldots, x_n; x_1^j, \ldots, x_n^j\rangle$ with $|x_1^j, \ldots, x_n^j\rangle$ as the control register and $|x_1, \ldots, x_n\rangle$ as the target register. Then, a *NOT* gate is applied to reverse the states in the first register:

$$|\psi_1\rangle = \prod_{s=1}^{n} NOT(x_s) CNOT(x_s^j, x_s) |\psi_0\rangle = \frac{1}{\sqrt{m}} \sum_{j=1}^{m} |d_1^j, \ldots, d_n^j; x_1^j, \ldots, x_n^j, y^j\rangle \tag{33}$$

If bits x_s and x_s^j are the same, then $d_s^j = 1$; otherwise $d_s^j = 0$. The reversing of the states in the first register is merely a mathematical trick. The Hamming distance between x^j and x is $d_{\text{Ham}}(x^j, x) = \sum_{s=0}^{n} (1 - d_s^j)$.

The fourth step is to compute the Hamming distance using $|d_1^j, d_2^j, \ldots, d_n^j\rangle$ and separate the set of training vectors that have a Hamming distance with the test vector of at most t, hence obtaining the following quantum state:

$$|\psi_2\rangle = \frac{1}{\sqrt{m}} \Big(\sum_{j \in \Omega} |d_1^j, \ldots, d_n^j; x_1^j, \ldots, x_n^j, y^j; 1\rangle + \sum_{j \notin \Omega} |d_1^j, \ldots, d_n^j; x_1^j, \ldots, x_n^j, y^j; 0\rangle \Big) \tag{34}$$

where $\Omega = \{j : d_{\text{Ham}}(x^j, x) \leqslant t\}$.

The incrementation circuit ($a = a + 1$) is used for computing the Hamming distance. Denoting this as Inc_{d_s}, the sum $\sum_{s=1}^{n} d_s^j$ can be obtained by invoking Inc_{d_s} n times. Let us add the third register $|a\rangle$ and the utility qubit $|u\rangle$. Initially, $|1\rangle$ is assigned to the last utility qubit. Recalling that the meaning d_s^j is reversed, such that the Hamming distance is less than or equal to t implies $\sum_{s=1}^{n} d_s^j \geqslant n - t$. With g chosen according to $2^{g-1} \leqslant n \leqslant 2^g$ (i.e., $g = \lceil \log_2 n \rceil$) and $l = 2^g - n$, then the condition that the Hamming distance be less than or equal to t can be rewritten as

$$\begin{cases} \sum_{s=1}^{n} d_s^j + l \geqslant n + l - t; \\ \sum_{s=1}^{n} d_s^j + l + t \geqslant 2^g \end{cases} \tag{35}$$

If the initial $a = l + t$, then the condition that the Hamming distance be less than or equal to t can be determined by whether the sum of $\sum_{s=1}^{n} d_s^j + a$ overflows or not. $g + 1$ qubits are required for getting $\sum_{s=1}^{n} d_s^j + a$. For checking the condition after getting a result of $\sum_{s=1}^{n} d_s^j + a$, the most significant qubit is selected and gets the "overflow check" signal, which indicates the condition that the Hamming distance be less than or equal to t.

The fifth step is to apply a partial measurement on the utility qubit $|u\rangle$. The probability of getting a 1-result

is $Pr_1 = |\Omega|/m$. After the measurement, we get the following state:

$$|\psi_3\rangle = \alpha \sum_{j\in\Omega} |d_1^j, \ldots, d_n^j; x_1^j, \ldots, x_n^j, y^j; a; 1\rangle,$$

$$\alpha = \frac{1}{\sqrt{m \cdot Pr_1}} \tag{36}$$

If the result of the measurement of $|u\rangle$ is $|1\rangle$, then the next step is to measure the "class register". After the measurement of the "class register" $|y\rangle$, the probability of obtaining class c is $Pr(c) = \sum_{j\in\{y^j=c\}} |\alpha|^2$. We obtain the class that has more vectors from Ω (the training vectors with a distance of at most t) with higher probability. The final classification result is an analog for "majority voting" in the classical NN algorithm.

If $|\Omega| = \sigma \cdot m$, $\sigma \in (0, 1]$, then $Pr_1 = \sigma$.

If $\omega{\cdot}\sigma{\cdot}m$ vectors among the above $\sigma{\cdot}m$ vectors belong to the c class (where $\omega \in (0, 1]$), then $Pr(c) = \omega\sigma$.

If we repeat the algorithm enough times, we will obtain the probability distribution, which will carry information about how close the members of each class are to the test vector.

The time complexity of the algorithm can be computed as follows:

$$\text{time}(Inc_{d_s}) = \begin{cases} 1, & \text{if } n == 1; \\ 10, & \text{if } n == 2; \\ 2n^2 + n - 5, & \text{if } n \geqslant 3 \end{cases} \tag{37}$$

We invoke Inc_{d_s} n times. So the total time complexity of the main stage of the algorithm is $O(n^3)$.

The algorithm is presented as Algorithm 5, and is labelled as RXLTL–NN(X, m, n, x, t), where X is a training set, m is the length of the training set, n is the length of each vector in the training set, x is the test vector, and t is a chosen parameter. If we obtain a 0-result from the utility qubit measurement, then we cannot obtain a class, and the algorithm returns -1, representing "unknown".

Algorithm 5 RXLTL-NN(X, m, n, x, t). Ruan-Xue-Liu-Tan-Li algorithm.

$|T\rangle \leftarrow$ Construct_Superposition(X, m, n) ▷ Construct a superposition of the training set X: $\frac{1}{\sqrt{m}} \sum_{j=1}^{m} |x_0^j, \ldots, x_{n-1}^j, y^j\rangle$
$|x\rangle \leftarrow |x_0, \ldots, x_{n-1}\rangle$ ▷ test vector
$g \leftarrow \lceil \log_2 n \rceil$
$l \leftarrow 2^g - n$
$a \leftarrow t + l$
$|a\rangle \leftarrow a$
$|u\rangle \leftarrow |1\rangle$
$|\psi\rangle \leftarrow |x; T; a; u\rangle$ ▷ initial state
$|\psi\rangle \leftarrow \prod_{s=0}^{n-1} NOT(x_s)CNOT(x_s^j, x_s)|\psi\rangle$
for $s \in \{0, \ldots, n-1\}$ **do**
  $|\psi\rangle \leftarrow 2CNOT(x_s, u, a_0)|\psi\rangle$
  for $r \in \{0, \ldots, n-2\}$ **do**
    $|\psi\rangle \leftarrow 2CNOT(x_s, a_r, u)|\psi\rangle$
    $|\psi\rangle \leftarrow 2CNOT(x_s, u, a_{r+1})|\psi\rangle$
  end for
  $|\psi\rangle \leftarrow NOT(u)|\psi\rangle$
end for
$ut \leftarrow$ Measurement$(|u\rangle)$
if ut is $|1\rangle$ **then**
  $c \leftarrow$ Measurement$(|y\rangle)$ ▷ measure register $|y\rangle$
end if
if ut is $|0\rangle$ **then**
  $c \leftarrow -1$
end if
return c

5 Quantum Nearest Neighbors Algorithm with Quadratic Speed-up

In this section, we present a classification method that is developed for non-binary classification problems. Wiebe et al.[11] suggested a quantum version of the NN algorithm with a quadratic speed-up. This quantum algorithm uses the minimum search algorithm[5] and subroutines for computing distances between vectors[11].

In the classical algorithm, the function F can be written as the k-NN algorithm for $k = 1$:

$$F(x) = y_j, \text{ for } j = \underset{i\in\{1,\ldots,m\}}{\text{argmin}} ||x^i - x|| \tag{38}$$

The classical algorithm is a linear search among vectors $x_1, \ldots, x_m$ for the minimum of $||x^i - x||$. The procedure for computing the distance between two vectors has complexity $O(n)$, so the algorithm has the following complexity.

Property 8. The classical query complexity of the k-NN algorithm is height(1NN) $= O(nm)$.

Quantum version of the nearest neighbor algorithm. First, the maximum search algorithm[5] is used; second, distance computing algorithms are used for computing the distance between vectors x^i and x. Both of these methods were described in the first part of the paper.

The assumptions are follows:

(1) The input vectors $x^1, \ldots, x^m$ and x are d-sparse for some $1 \leqslant d \leqslant n$. In other words, each vector x^i has only d non-zero elements.

(2) Quantum oracles are provided in the form

$$\mathcal{O}|i\rangle|j\rangle|0\rangle \to |i\rangle|j\rangle|x_j^i\rangle,$$
$$\mathcal{F}|i\rangle|l\rangle \to |i\rangle|\text{pos}(i,l)\rangle,$$

where $\text{pos}(i,l)$ is the position of the l-th non-zero element of x^i.

(3) There is an $r_{\max}$ such that $0 \leqslant x_j^i \leqslant r_{\max}$ for any $i \in \{1,\ldots,m\}$, $j \in \{1,\ldots,n\}$.

(4) Each vector is normalized to 1, for convenience.

It is known that for any two vectors a and b, the following equality holds:

$$\|a-b\|^2 = \|a\|^2 + \|b\|^2 - 2(a\cdot b) \tag{39}$$

where $a\cdot b$ is the inner product of a and b. Therefore, the following property holds:

$$F(x) = y_j, \text{ for } j = \underset{i\in\{1,\ldots,m\}}{\text{argmin}} ||x^i - x|| = \underset{i\in\{1,\ldots,m\}}{\text{argmax}} (x^i\cdot x)^2 \tag{40}$$

The quantum algorithm is presented as Algorithm 6.

The algorithm has the complexity given in the following property.

Property 9. The quantum query complexity of the quantum version of the nearest neighbor algorithm is $O(\sqrt{m}\log m\cdot d^2 r_{\max}^4)$.

Comments. As already discussed, the algorithm works for d-sparse vectors, where $1 \leqslant d \leqslant n$. This means that in the general case, we realize the speed-up if $\sqrt{m} > n$. This setup is reasonable for a range of problems, as problems with d-sparse vectors in training and testing sets are not unusual.

Another restriction of the algorithm is $r_{\max}$. Typically, a prepossessing procedure can be performed that normalizes data and changes $r_{\max}$. The complexity of this procedure is $O(mn)$, but it will be performed only once, so we do not take account of the complexity of the peprocessing procedure in the case of a large test set.

6 Conclusion

This paper surveys quantum algorithms for binary classification and discusses the quantum nearest neighbor algorithms. It further analyzes the quantum nearest neighbor algorithms and reveals their quadratic speed-up over classical algorithms.

Algorithm 6 QNN(x, (x^1,…,x^m)). Quantum version of the nearest neighbor algorithm.

$|x\rangle \leftarrow x$
$j \leftarrow$ Max_inner_product($|x\rangle$, ($|x^1\rangle$,…,$|x^m\rangle$))
return y^j

Acknowledgment

This work was supported in part by the Russian Science Foundation (No. 19-19-00656) and the Natural Science Foundation of Guangdong Province, China (No. 2019A1515011721).

References

[1] C. Cortes and V. Vapnik, Support-vector networks, *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[2] F. Rosenblatt, The perceptron: A probabilistic model for information storage and organization in the brain, *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

[3] J. A. K. Suykens and J. Vandewalle, Least squares support vector machine classifiers, *Neural Processing Letters*, vol. 9, no. 3, pp. 293–300, 1999.

[4] D. Anguita, S. Ridella, F. Rivieccio, and R. Zunino, Quantum optimization for training support vector machines, *Neural Networks*, vol. 16, nos. 5&6, pp. 763–770, 2003.

[5] C. Durr and P. Høyer, A quantum algorithm for finding the minimum, arXiv preprint arXiv: quant-ph/9607014, 1996.

[6] L. K. Grover, A fast quantum mechanical algorithm for database search, in *ACM Symp. on Theory of Computing*, Philadelphia, PA, USA, 1996, pp. 212–219.

[7] A. Kapoor, N. Wiebe, and K. Svore, Quantum perceptron models, in *Advances in Neural Information Processing Systems*, Barcelona, Spain, 2016, pp. 3999–4007.

[8] N. Wiebe, A. Kapoor, and K. M. Svore, Quantum perceptron models, in *Neural Information Processing Systems*, Barcelona, Spain, 2016, pp. 4006–4014.

[9] M. Schuld, I. Sinayskiy, and F. Petruccione, Quantum computing for pattern classification, in *Pacific Rim International Conference on Artificial Intelligence*, Springer, 2014, pp. 208–220.

[10] Y. Ruan, X. L. Xue, H. Liu, J. N. Tan, and X. Li, Quantum algorithm for k-nearest neighbors classification based on the metric of hamming distance, *International Journal of Theoretical Physics*, vol. 56, no. 11, pp. 3496–3507, 2017.

[11] N. Wiebe, A. Kapoor, and K. M. Svore, Quantum algorithms for nearest-neighbor methods for supervised and unsupervised learning, *Quantum Information & Computation*, vol. 15, nos. 3&4, pp. 316–356, 2015.

[12] T. Cover and P. Hart, Nearest neighbor pattern classification, *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[13] K. Fukunaga and P. M. Narendra, A branch and bound algorithm for computing k-nearest neighbors, *IEEE Transactions on Computers*, vol. C-24, no. 7, pp. 750–753, 1975.

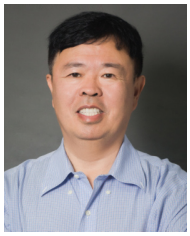
[14] P. Rebentrost, M. Mohseni, and S. Lloyd, Quantum support vector machine for big data classification, *Physical Review Letters*, vol. 113, no. 13, p. 130503, 2014.

[15] C. A. Trugenberger, Probabilistic quantum memories, *Physical Review Letters*, vol. 87, no. 6, p. 067901, 2001.

[16] C. A. Trugenberger, Quantum pattern recognition, *Quantum Information Processing*, vol. 1, no. 6, pp. 471–493, 2002.

Farid Ablayev received the habilitation degree (doctor of physics and mathematics-second level after the PhD level) at Moscow State University. He is a professor in Kazan Federal University and Kazan E. K. Zavoisky Physical-Technical Institute. His current research interests include complexity theory, quantum computing, automata theory, machine learning, and data stream processing algorithms.

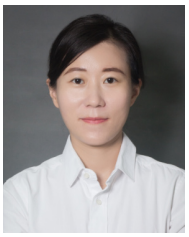
Marat Ablayev received the master degree from Kazan Federal University in 2005. He is a researcher in Kazan Federal University and Kazan E. K. Zavoisky Physical-Technical Institute. His current research interests include complexity theory, quantum computing, automata theory, machine learning, and data stream processing algorithms.



Joshua Zhexue Huang received the PhD degree from the Royal Institute of Technology, Sweden. He is now a professor in the College of Computer Science and Software, Shenzhen University, and professor and chief scientist in the Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, and honorary professor in the Department of Mathematics, the University of Hong Kong. His research interests include data mining, machine learning, and clustering algorithms.

Kamil Khadiev received the PhD degree in 2015. He worked in Institute of Informatics of Tatarstan Academy of Science, University of Latvia, Smart Quantum Technologies Ltd, Kazan E. K. Zavoisky Physical-Technical Institute, Kazan Federal University. His current research interests include quantum computing, quantum algorithms, communicational complexity, automata theory, branching programs, machine learning, and data stream processing algorithms.

Nailya Salikhova is a PhD student in Kazan Federal University. Her current research interests include quantum computing, machine learning, and data stream processing algorithms.



Dingming Wu received the PhD degree in computer science in 2011 from Aalborg University, Denmark. She is an assistant professor with College of Computer Science & Software Engineering, Shenzhen University, China. Her general research area is data management and mining, including data modeling, database design, and query languages, efficient query and update processing, indexing, and mining algorithms.