Radboud University Nijmegen

Departement of High Energy Physics and ING Quantum R&D

Master's thesis

Building a quantum kNN classifier with Qiskit: theoretical gains put to practice

Author

D.J. Kok
s4467205

Supervisors

Radboud University
Dr. S. Caron
S.M.M. Otten MSc.

ING
Dr. A. Acun

February, 2021

Preface

This thesis, “Building a quantum kNN classifier with Qiskit: theoretical gains put to practice”, is a project for building a quantum algorithm capable of performing kNN classification. It has been written as completion of the Master’s course Particle- and Astrophysics at the Radboud University of Nijmegen. From February 2020 to February 2021 I was working on this thesis by investigating quantum computing, programming, and documenting my findings.

This thesis is a collaborative project between the High Energy Physics (HEP) department at the Radboud University and the Quantum Research group at ING. I established my goal of this thesis together with Sascha Caron and Sydney Otten from HEP, and Adil Acun from ING. These were also my supervisors during this project. I have learned a lot during my internship at ING and the HEP department, e.g. how to work with a quantum computing programming language; reading and retrieving information directly from code documentation or source code; and how to properly build up, publish and maintain a Python package from scratch. It was a special experience to be an intern during a pandemic, which limited my experience at a corporation and the number of connections that I had hoped to make.

I want to thank Sascha Caron and Adil Acun for giving me the opportunity to perform my internship at the HEP department and ING. I would also like to thank my girlfriend, Manon de Jonge, for her incredible support during good and bad times. This year was because of numerous reasons a physically and mentally challenging period and you were there for me. Furthermore, I would like to thank Sydney Otten and Adil Acun for their impressive guidance and support during my internship. Your patience and acceptance of my periods when no work was done, followed by your advice and nudges in the right direction kept me at work, and always made me get back on my feet again. Moreover, I want to thank my (ex-)roommates for the close relations that I had with you during this pandemic, sculpting the life we had before the pandemic into our day-to-day life during it. Because of you, I had no real struggle with the lockdown or guest limit since my friends were always there. Finally, I want to thank my family for their interest in my work, even though it is not their expertise. You provided a listening ear to my stories and experiences. Thank you all.

Daniël J. Kok, Nijmegen, Thursday 21st, January 2021.

Abstract

The use cases of and adaptation on machine learning increase as its applicability rises. One of the adaptations is the translation of machine learning algorithms from classical computers to quantum computers, since the hardware and accessibility of quantum computing is increasing as well. Quantum computing theoretically reduces the resource complexity of classical methods exponentially because of the nature of superposition, promising reduced runtimes on larger data sets in machine learning. The kNN is a simple and robust method and is translated to a pure quantum algorithm (QkNN) to perform classification on three data sets: the Iris data, provided by `scikit-learn`, because of its familiarity, and two data sets related to physics and finance. Translating this to a quantum algorithm promises a complexity of $\log_2(ND)k$ (instead of NDk for classical methods) for determining k nearest neighbors in a data set with N vectors with dimensionality D. The code in this thesis, including tutorials and documentation, are available on PyPI, ReadTheDocs, and GitHub.

The QkNN is developed within Qiskit and distance measurement is done by measuring the fidelity $F = \|\langle\psi|\phi\rangle\|^2$ between to quantum states ψ, ϕ, defining the distance as $D = 1 - F$. The fidelity can be measured using a $SWAP$-test quantum circuit and an Oracle $\mathcal{W}$ is produced to measure the fidelity of a full data set in superposition instead of one vector at a time. The kNN from `scikit-learn` is used and configured to use the cosine similarity $(\boldsymbol{x}, \boldsymbol{y})$ as a replacement for the fidelity F. The data are also encoded using the analog (amplitude) encoding method before loaded onto the quantum circuit.

The results show that the QkNN performs similar, although often a bit worse, than the kNN. There is also a constant variability in the accuracies of the QkNN for the finance and physics data. However, the variability in accuracy for the Iris data reduces as the number of shots of the `qasm_simulator` increases. Furthermore, the runtime as function of complexity $\log_2(ND)$ show a quadratic increase for the QkNN against an almost constant runtime for the kNN.

The present-day use of the QkNN is not advised until the accessibility to the SDK increases, the noise on a quantum computer reduces, and algorithms of larger scale can run better.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The goal of this thesis is to write a pure quantum model to perform a k-nearest neighbors (kNN) classification on provided, pre-classified, data. The second goal is to perform benchmarks on the required algorithm concerning its classical counterpart and to critically review the results. The last but also important goal is to make the resulting algorithm distributable and as accessible as possible, so that further development on this work is easy and approachable. Next to this, it is an attempt to prevent it from being buried in the large pile of science (as is often the case). Hence, the code for the algorithm has been published on PyPI for people to access and install on their system for ease of access (Kok 2020a). To help with understanding the code, a full documentation page is brought to life, hosted by Read the Docs (Kok 2020b), and a GitHub page exists for any voluntary contribution (Kok 2020c).

This thesis is set up as follows: an introduction (this chapter) to the current state of machine learning and its application within the financial and scientific sector in section 1.1. It is followed up by the introduction of the kNN in section 1.2. Finally, the adaptation to quantum computing and what role it has within this thesis is explained in 1.3, which introduces the quantum kNN (QkNN).

After this, chapter 2 describes the method of working out the QkNN and how the benchmarks are performed. Subsequently, the results of the built circuit in chapter 2 are shown in chapter 3 and analyzed and reviewed in chapter 4. Afterward, the conclusion is drawn in chapter 5.

1.1 Machine learning in physics and finance

Machine learning is a popular and a widely studied field, especially since the last decade. It often performs many small calculations on large amounts of data to determine statistical outcomes. This means that the groups which heavily benefit from these methods are often those with large amounts of data that need to be analyzed. The problem with the popularity of machine learning is that more and more groups want to participate even though their goal does not initially lie within the field of big data.

A section in Institute of Electrical and Engineers (2017) is dedicated to the usefulness of machine learning in security devices but the information found in this work can be spread out to almost every field. A direct quote from the paper is:

> "...the effectiveness of the algorithms is directly proportional to the quantity and quality of the data" – Institute of Electrical and Engineers

The relation between the effectiveness of data science in direct relation to the quality and quantity of the data is something that is often forgotten in the hype of machine learning. The first half of this quote describes that too little or too much data can negatively influence the process that it is trying to aid, and finding out what is "too little" or "too many" is a subject in itself. The second half of the quote points out the relation that data have to the process that they are trying to aid, e.g. for predicting the chance of rain, the wind direction is better quality data than the number of people in the city center. Something important that needs to be added to this is that the most important step in data science is the pre-processing of the data: how does the data look, are there any (unexpected) outliers, and should the data be manipulated such that all parameters have equal participation in the task? These principles are apparent in both physics and finance, and the next paragraphs will describe the application of machine learning in these fields.

1.1.1 Physics and High Energy Physics

The first and foremost application of machine learning for the completion of this thesis is within the field of physics. Machine learning is applied in a lot of fields at the Radboud University, one of which is the department of High Energy Physics (HEP).

Since physics as a subject has a solid theoretical foundation that describes events in high detail which, in turn, can be observed in practical physics, a lot of data is present to perform science with. Therefore, the quality of the data is generally very high. On the other hand, the amount of data is very large. Considering this, it is often the goal to distinguish higher quality data from lower quality and reduce their dimensionality.

That is why Diblen et al. (2020) have produced an interactive web application to help with visualizing large amounts of data interactively. Examples of direct applications within machine learning are Caron, Heskes, et al. (2019), which apply active learning to reduce the parameter space of data, Gucht et al. (2020) to apply deep learning to recover accreting black hole parameters from imaging, and, as a final example, Caron, Gómez-Vargas, et al. (2018) have produced a convolutional neural network to analyze excessive emission of γ-rays in the galactic center.

To summarize, whilst machine learning is already widely applied at the HEP department at Radboud University, quantum computing has yet to make its introduction. The reason of interest in this area for HEP is the possibility of reducing computation time of performing classification tasks on large amounts of data with lower dimensionality. For example, the European Organization for Nuclear Research (CERN) provides particle accelerators for HEP research by collision events. These events generate a large amount of data in very short timeframes. An application of kNN classification could be applied here to filter noise in big data classification (García-Gil et al. 2019). Quantum computing promises an exponential speed-up on big data (Montanaro 2015) to an extent where noise filtering could be applied real-time during data gathering of CERN collision events. Another goal would be to perform density determination (Zhao and Lai 2020) on CERN collision data to quantify outliers in experimental results. A quantum algorithm could outclass a classical algorithm because of the high amount of data from CERN.

1.1.2 Banking and OrangeQ

In finance, however, some work has already been done within the field of quantum computing. Within Internationale Nederlanden Groep (ING), the company where this thesis is part of, a select group of people is dedicated to the research and development (R&D) of the possibilities of applying quantum computing and machine learning. This group is lead by Dr. Adil Acun and, albeit still in development, called OrangeQ. The purpose of OrangeQ is to enable the bank and its customers to keep up with the digital and innovative transformation that the world is undergoing.

Many survey papers are discussing the potential use of applying data science within banking (Heaton et al. 2018, Ghoddusi et al. 2019, Rundo et al. 2019). These surveys aid in overcoming the difficulty of filtering out bad quality data from all data but, in addition, Wittenbach et al. (2020) goes into depth on the challenges and opportunities with financial data in particular. Some practical examples of how the financial sector has tried applying the machine learning methods are: risk analysis (Bracke et al. 2019), product pricing (Gan et al. 2020) or customer retention (Kumar et al. 2021).

There also are some applications of quantum machine learning in finance. Some examples are: the use of Monte Carlo simulations for risk analysis (Woerner and Egger 2019), combinatorial optimization for optimal trade opportunities (Rosenberg et al. 2016) and quantum deep learning for option pricing (Zoufal et al. 2019). The goal of OrangeQ is to combine quantum computing and machine learning to make

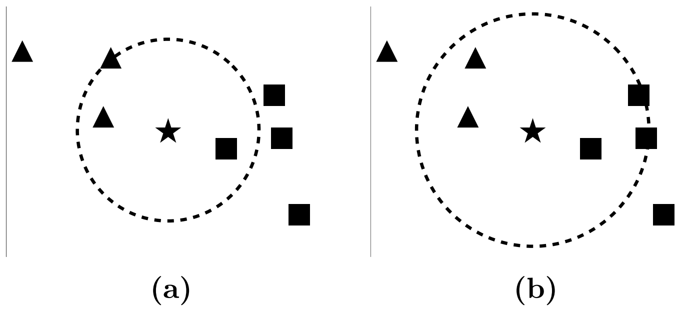


Figure 1.1: Schematic representation of the kNN algorithm in two dimensions. An unclassified data point (star) is added in the binary data set indicated by triangles and squares. The range of the amount of nearest neighbors used for classification is set to 3 (a) and 5 (b) and is represented by the dashed line.

an impact on finance and banking.

1.2 The k-nearest neighbors algorithm

This thesis will dive into the workings of a basic machine learning algorithm called the k-nearest neighbors (kNN) algorithm. The kNN is at its foundation a basic yet powerful method for data classification. In a nutshell, it works as follows:

1. determine the distance of *all* N points with dimension D to an unclassified data point;
2. receive the k points closest to the data point;
3. do a majority vote to identify the class of the data point.

A schematic example of how this algorithm operates is shown in figure 1.1. It shows an unclassified datapoint, represented by a star, which needs to be classified using the kNN algorithm on a binary classified, two dimensional dataset. The classes of the dataset are indicated by either a triangle or a square. To visualize the data partaking in the majority vote, a circle is drawn using the distance of the furthest included neighbor as the radius. Figure 1.1a shows the kNN for $k = 3$ nearest neighbors, where the unclassified data point should be classified as a triangle. Figure 1.1b, however, shows what would happen for $k = 5$: two more squares would partake in the majority vote, causing the class of the star to be set to a square. The value which k should take is a known problem and a research question on itself (Zhang et al. 2017), and there is no real optimal value that works for every dataset, albeit a heavily developed subject with promising results of adaptive

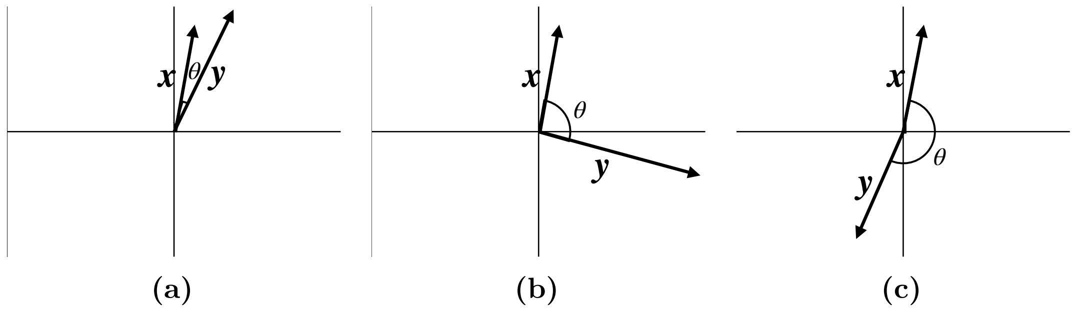


Figure 1.2: Some exemplary two-dimensional vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ to compute the cosine similarity of, with the angle θ the angle between $\boldsymbol{x}$ and $\boldsymbol{y}$. (a): two almost overlapping vectors with $\theta \approx 0\,\text{deg}$. (b): two orthogonal vectors with $\theta \approx 90\,\text{deg}$. (c): two opposite vectors with $\theta \approx 180\,\text{deg}$.

algorithms (Bian et al. 2020). It can be considered as a hyperparameter and the most optimal value for the data used in this thesis can be found out via learning. This is, however, out of the scope for this thesis.

To return to the basics of the kNN, the distance determination (described in step 1 of the three steps above) requires a form of distance measurement. This is usually done by handling the space as a Euclidian space and using the Euclidian metric for distance calculation. There are, however, different methods for distance calculation. The one that suits this case the best and coincides the most with the QkNN (see section 1.3.3), uses the cosine similarity as distance measurement:

$$(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x}\boldsymbol{y}}{\|\boldsymbol{x}\|\|\boldsymbol{y}\|}. \tag{1.1}$$

This equation will take a value between -1 and 1, or, in the quantum computational case, 0 and 1, since that only deals with positive space. $(\boldsymbol{x}, \boldsymbol{y})$ is an indication of how much the vectors x and y overlap with each other.

Figure 1.2 shows some two-dimensional exemplary vectors $\boldsymbol{x}, \boldsymbol{y}$ to which $(\boldsymbol{x}, \boldsymbol{y})$ would hold different values. A full overlap (1.2a) means that the angle θ between the two vectors is zero, and coincides with a cosine s imilarity $(\boldsymbol{x}, \boldsymbol{y}) = 0$. If $\theta = 180$ degrees (1.2c), the vectors are complete opposite to each other and $(\boldsymbol{x}, \boldsymbol{y}) = 1$. The in-between is shown in 1.2b, where two orthogonal vectors produce $\theta = 90$ degrees, and $(\boldsymbol{x}, \boldsymbol{y}) = 0.5$. Hence, the "distance" D can be determined via $1 - (\boldsymbol{x}, \boldsymbol{y})$.

Secondly, the algorithm chooses the k data points with the lowest distance D from the uncategorized data point. This is finally followed by the third step, where the k nearest neighbors to the data point participate in a voting session to determine the class of the unclassified point.

This third step is specific for the kNN *classifier*. There is also a kNN *regressor*, where instead of a majority vote at step 3, the algorithm takes the average of the k

nearest neighbors. This thesis concerns the classification case only. Since it is a thoroughly studied algorithm, the kNN from the `scikit-learn` library (Pedregosa et al. 2011) will be used. It will be carefully parametrized so that the comparison between the kNN developed QkNN will be as direct as possible.

1.3 Quantum computing

Before the QkNN can be explained entirely, the subject of quantum computing needs to be introduced. Quantum computing is similar to probabilistic programming in such a way that it uses matrix manipulations describing time evolution applied to a state. There is an important difference, however, since quantum computing uses complex values to describe probabilistic states. This means that the matrices describing time evolution can *interfere* with each other, canceling each other out instead of simply adjusting the probabilistic outcome (what would happen in the classical probabilistic case). For more info, see Yanofsky (2008).

The theoretical model of a quantum computer was first developed in the 1980 when Benioff developed a quantum mechanical counterpart to the Turing machine (Benioff 1980). Feynman followed this up with his description of what kind of simulation one could run using said quantum computer in 1982 (Feynman 1982). Probabilistic states on classical machines are always approximations on a classical machine because these need to be translated to the bits in 0's and 1's. Feynman stated that one could benefit from the probabilistic nature of quantum matter to exactly embed a probabilistic state on a quantum computer. Furthermore, Deutsch did similar work to Benioff, which helped lay the foundation of theoretical quantum computing (Deutsch 1985).

The method of quantum computing then developed in multiple ways, where the most common method is that of the quantum circuit. A quantum circuit is built by using quantum bits or qubits for short. These qubits work as regular bits, where they can be either a 0 or a 1, but can also be brought in a probabilistic state as described by Feynman (1982), and take a superposition between 0 and 1. The result of a quantum circuit is then a probabilistic state in itself and can only be determined by running the circuit multiple times. This kind of quantum computer is also referred to as a *gated quantum computer* since the qubits can be manipulated into certain states with the help of quantum logic gates.

Feynman put into words the essential nature of the workings of a quantum computer. Instead of performing calculations by performing step-by-step operations, the physical problem itself is embedded into the hardware to do the calculation. The probabilistic nature of the particles carry the information of the physical problem through the quantum manipulations performed within the quantum computer. Technically, there is no computation involved here.

Two known companies to create a gated quantum computer are the International Business Machines Corporation (IBM) and Google. There is also a second line of quantum computer (not considering those that have not yet been build), called the *quantum annealer*, which uses a large system of qubits brought to a superposition to solve one problem using the annealing method (Boixo et al. 2014).

IBM and Google have both set their goals to release a software development kit (SDK) to write and create quantum circuits to run on their hardware. Because of the closer relations to IBM and its larger community (at the time of writing), the SDK for their quantum computer is used, called Qiskit (Aleksandrowicz et al. 2019).

Qiskit

This software package is open source and allows users to access a couple of qubits to run small circuits. A good introduction to working with Qiskit is also provided on their website but a small example is shown here to emphasize the ease of basic quantum computing.

A quantum circuit exists of quantum bits and can be created by:

```python
import qiskit

circ = qiskit.QuantumCircuit(3)
```

which creates a quantum circuit with 3 qubits. It can be expanded by adding gates:

```python
# Add an H gate on qubit 0, putting this qubit in superposition.
circ.h(0)
# Add a CX (CNOT) gate on control qubit 0 and target qubit 1,
# putting the qubits in a Bell state.
circ.cx(0, 1)
# Add a CX (CNOT) gate on control qubit 0 and target qubit 2,
# putting the qubits in a GHZ state.
circ.cx(0, 2)
```

The entire quantum circuit can be drawn by using matplotlib (indicated by the `'mpl'` argument):

```python
circ.draw('mpl')
```

to produce a figure as shown in figure 1.3 (p. 8). This is the most common method of displaying a quantum circuit, with the labels of the qubits displayed on the left. Each qubit has a horizontal line, called a *quantum wire*, that displays the

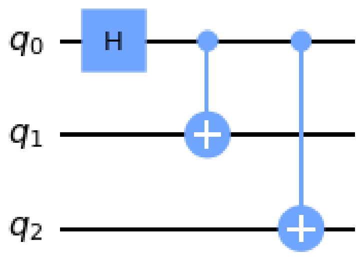
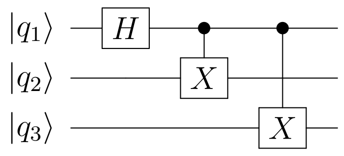


Figure 1.3: An exemplary quantum circuit drawn by Qiskit corresponding to the tutorial on their website. The qubits in this quantum circuit are indicated by q_i. The horizontal lines represent the time evolution of the qubits and the gates are indicated in blue. The H being the Hadamard gate, and the $+$-filled circles are the X-gate controlled by q_0 indicated by a blue dot.

time evolution of the qubit in the circuit. Gates that are applied on the qubit are displayed in a box with a label indicating the type of gate. In figure 1.3, the gates applied are the Hadamard (H) and controlled X (cX) gate. The H-gate acts on a single qubit, whereas the cX-gate acts on two qubits at once. This same circuit can also be displayed in a mathematical equation which will be the main method to display quantum circuits from now on:



(1.2)

This quantum circuit can now be run by using IBM's quantum computer online or by simulating a quantum computer locally.[1] Qiskit provides the simulation in their *Aer* backend, where there are multiple simulators. The `statevector_simulator` can return the exact state of all n qubits in a quantum circuit. Furthermore, the `unitary_simulator` creates the full $2^n \times 2^n$ matrix (for n qubits) representing the result of all the unitary operations on the qubits. Finally, there is the `qasm_simulator`, which simulates the execution of the quantum circuit on a quantum computer, and thus the collapse of the quantum state embedded in the quantum circuit (represented by the state vector from `statevector_simulator`), into an observable result. This means that when the quantum circuit from figure 1.3 is observed once, the result will be a combination of 0's and 1's, e.g. $|0\rangle|0\rangle|1\rangle$. To

[1]or a local quantum computer, should one have access to one.

measure the full quantum state of the three qubits, the measurement needs to be repeated multiple times. For further details on how to work with these simulators, IBM provides a thorough tutorial on their website.

1.3.1 Quantum circuit operations

To explain how the manipulation of quantum matter is implemented on a quantum computer, the inner workings of a qubit and its representation must be explained. A qubit q is represented in Bra-ket notation, i.e. $\langle q|$, (the Bra) and $|q\rangle$ (the Ket), or together as a Bra-ket: $\langle q|q\rangle$. A qubit q can be brought to a certain state existing of two states: the $|0\rangle$ and $|1\rangle$. These kind of states must be described by two orthogonal vectors, commonly done via:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \tag{1.3}$$

Note that these states are described by Kets: the Bra-version of these states act in this case as the complex conjugate, e.g.

$$\langle 0| = |0\rangle^{\dagger} = \begin{bmatrix} 1 & 0 \end{bmatrix}, \tag{1.4}$$

which, in the case of only real-valued vectors, is the transposed vector. The qubit q is described by a combination of the $|0\rangle$ and $|1\rangle$, indicated by two complex amplitudes $\alpha, \beta \in \mathbb{C}$:

$$|q\rangle = \alpha|0\rangle + \beta|1\rangle. \tag{1.5}$$

The amplitudes α, β of the states $|0\rangle, |1\rangle$ describe the probability of $|q\rangle$ collapsing into that state once it is measured. By way of clarification, the Bloch sphere (figure 1.4) is introduced to visualize the state of a single qubit.

In this figure, a qubit state $|\psi\rangle$ is represented by a line pointing in a certain direction indicated in spherical coordinates $(1, \theta, \phi)$ with on the sphere. A qubit represented in a Bloch sphere would be in state $|0\rangle$ $(\alpha = 1, \beta = 0)$ if the line is pointing to the north $(+\hat{z}, \theta = 0\,\text{deg})$ of the sphere and in state $|1\rangle$ $(\alpha = 0, \beta = 1)$ if the line is pointing to the south $(-\hat{z}, \theta = 180\,\text{deg})$ of the sphere. This shows that the angle θ indicates the *state* of the qubit. The angle ϕ indicates the *phase* of the qubit. For example: a phase change changes the phase of a qubit from $|1\rangle$ to $-|1\rangle$, but a state change changes a $|1\rangle$ to $|0\rangle$.

The probability of measuring the qubit in e.g. state $|1\rangle$ can be determined by

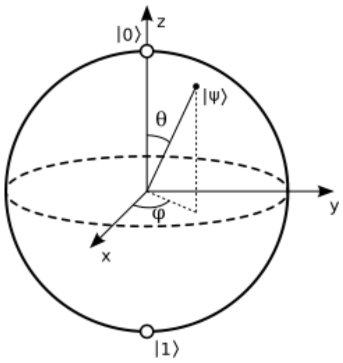


Figure 1.4: A representation of the Bloch Sphere, used for describing qubit states. The zx-plane and xy-plane show the real and complex values of the quantum state, respectively. An example state $|\psi\rangle$ is added to show that a quantum state is defined by its angles θ and ϕ w.r.t. the z and x axis, respectively. (Meister 2009).

calculating:

$$\begin{aligned}\langle q|1\rangle|q\rangle &= |q\rangle^\dagger \cdot |1\rangle \cdot |q\rangle \\ &= \begin{pmatrix}\alpha^* & \beta^*\end{pmatrix} \cdot \begin{pmatrix}0\\1\end{pmatrix} \cdot \begin{pmatrix}\alpha\\\beta\end{pmatrix} \\ &= \begin{pmatrix}\alpha^* & \beta^*\end{pmatrix} \cdot \begin{pmatrix}0\\\beta\end{pmatrix} \\ &= 0 + \beta^*\beta \\ &= |\beta|^2.\end{aligned} \tag{1.6}$$

In a like manner, $|\alpha|^2$ would be the probability of finding $|q\rangle = |0\rangle$. Since these two states are the only states that $|q\rangle$ can be in, this means that $|\alpha|^2 + |\beta|^2 = 1$, since the sum of all probabilities must be 1.

Multiple qubits can now be used to create a quantum *registers*. The main use of quantum registers is to create an indication that one group of qubits describe one variable. For example: to describe the number 6 in binary one would need three bits as $|q_1\rangle|q_2\rangle|q_3\rangle = |1\rangle|1\rangle|0\rangle$. These three bits can be gathered to a register $r = |110\rangle$ for clarity. These quantum registers can be mathematically represented via the tensor product, e.g. for two qubits $|q_1\rangle, |q_2\rangle$ forming the quantum register $|q_1q_2\rangle$:

$$|q_1q_2\rangle = |q_1\rangle \otimes |q_2\rangle = \begin{bmatrix}\alpha_{q_1}\\\beta_{q_1}\end{bmatrix} \otimes \begin{bmatrix}\alpha_{q_2}\\\beta_{q_2}\end{bmatrix} = \begin{bmatrix}\alpha_{q_1} \otimes \begin{bmatrix}\alpha_{q_2}\\\beta_{q_2}\end{bmatrix} \\ \beta_{q_1} \otimes \begin{bmatrix}\alpha_{q_2}\\\beta_{q_2}\end{bmatrix}\end{bmatrix} = \begin{bmatrix}\alpha_{q_1}\alpha_{q_2}\\\alpha_{q_1}\beta_{q_2}\\\beta_{q_1}\alpha_{q_2}\\\beta_{q_1}\beta_{q_2}\end{bmatrix}, \tag{1.7}$$

where $\alpha_{q_1}, \beta_{q_1}$ and $\alpha_{q_2}, \beta_{q_2}$ are the amplitudes of $|q_1\rangle, |q_2\rangle$ according to (1.5), respectively. This shows that all gated quantum computing has tensor calculus at its base, where the qubits are represented by vectors and can be brought into different states using matrices. Hence, tensor calculus can be applied to simulate the execution of a quantum circuit. This works fine for smaller simulations, but can become a very complex and expensive task for more complex circuits.

To build a quantum circuit performing a certain task, registers are initialized and manipulated into different states. The matrices used to manipulate qubits are called the *quantum gates*. A quantum algorithm is a series of manipulations on qubits with these quantum gates. Some common quantum gates (also used in this project) are the X-gate (often called the NOT-gate), the Hadamard- (or H) gate, and the $SWAP$-gate. The matrix representations of these gates are:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad SWAP = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{1.8}$$

When a gate is applied to a qubit, it is viewed as an operator acting on a quantum state. To illustrate, an operator $\hat{A}$ acting on a quantum state ϕ can be written as $\hat{A}|\phi\rangle$. For the X-gate, this is written as:

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle. \tag{1.9}$$

This shows that the X-gate flips the state of the qubit from $|0\rangle$ or $|1\rangle$ to respectively $|1\rangle$ or $|0\rangle$. Some examples of how the other gates shown in equation 1.8 perform are (without the worked-out tensor calculus):

$$H|0\rangle = \frac{1}{\sqrt{2}} \left(|0\rangle + |1\rangle \right), \quad SWAP|01\rangle = |10\rangle. \tag{1.10}$$

Furthermore, a set of gates called the *controlled* gates are gates which always act on at least two qubits, with at least one of the qubits acting as the controller and at least one being manipulated by the controlled action. The gate performs a certain action only if the controlling qubit(s) is (are) $|1\rangle$. By way of example, the controlled X (cX) gate is shown:

$$cX = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \tag{1.11}$$

which performs the X operation on the second qubit if the first qubit is $|1\rangle$, i.e. $cX|11\rangle = |10\rangle$, but $cX|01\rangle = |01\rangle$. This means that these two qubits are now *entangled* with each other, since the result of the second qubit depends on the state of the first. Such controlled operations are a powerful way of manipulating qubits only if a register is in a certain desired value, and use quantum entanglement to their benefit.

Finally, another set of important but more abstract gates are the so-called *rotation* gates:

$$R_x = \begin{pmatrix} \cos\frac{\theta_1}{2} & -i\sin\frac{\theta_1}{2} \\ -i\sin\frac{\theta_1}{2} & \cos\frac{\theta_1}{2} \end{pmatrix}, R_y = \begin{pmatrix} \cos\frac{\theta_2}{2} & -\sin\frac{\theta_2}{2} \\ \sin\frac{\theta_2}{2} & \cos\frac{\theta_2}{2} \end{pmatrix}, R_z = \begin{pmatrix} 1 & 0 \\ 0 & \exp i\theta_3 \end{pmatrix}. \tag{1.12}$$

These gates are parametrized gates described by a parameter angle θ_i, or all gates at once using a vector $\vec{\theta} = (\theta_1, \theta_2, \theta_3)$. Looking back at the Bloch sphere in figure 1.4 (p. 10),these θ_i are the angles between the yz-, xz- and xy-projection of $|\psi\rangle$ to the z-, x- and y- axis, respectively. Hence, the rotation gates R_x, R_y and R_z rotate the exemplary phase $|\psi\rangle$ around the x, y or z axis with an angle θ_i, respectively.[2]

The reason these gates are important is that it gives more freedom to create a certain quantum state on a qubit. The three gates from (1.8) create only a fixed manipulation and result. These results can be described on the zy-plane of the Bloch sphere: $|0\rangle$ and $|1\rangle$ as already explained previously, and in $\frac{1}{\sqrt{2}}(|0\rangle \pm |1\rangle)$ the arrow would be pointing in $\hat{y}$ $(+)$ or $-\hat{y}$ $(-)$. The rotation gates from (1.12) can create an arbitrary state pointing to arbitrary directions on the Bloch sphere. The parameter angles $\vec{\theta}$ are then translated to a probabilistic amplitude α or β according to (1.5).

An important thing to note is that all gates (or operators) must be hermitian, i.e. $H^\dagger = H$. If this were not the case, a quantum circuit could be constructed where energy is not conserved, resulting in non-physical solutions.

1.3.2 Quantum computing advantages and drawbacks

Using the methods from the previous section would create advantages in computing. These advantages, however, are accompanied by some drawbacks.

Firstly, the possible gains of translating currently known models to a quantum computer comes from the quantum nature of superposition, entanglement, and interference: *superposition* provides the method of saving 2^n numbers on n qubits, reducing the number of bits required exponentially (Lloyd, Mohseni, et al. 2013, Rebentrost et al. 2014 and Wiebe et al. 2014). A quantum register of n qubits has 2^n orthogonal states with amplitude $\alpha_i \in \mathbb{C}$ for $i \in [1, 2^n]$. For example,

[2]This means that θ in fig. 1.4 is a different parameter than $\vec{\theta}$ in (1.12). θ in figure 1.4 is from the spherical coordinate system, and $\vec{\theta}$ are the parameters to rotate $|\psi\rangle$ around a desired axis.

using equation 1.7, a register with 3 qubits would look like:

$$|q_1q_2q_3\rangle = \alpha_1 \begin{bmatrix}1\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} + \alpha_1 \begin{bmatrix}0\\1\\0\\0\\0\\0\\0\\0\end{bmatrix} + \ldots + \alpha_8 \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\1\end{bmatrix}, \tag{1.13}$$

and each amplitude α_i can hold one classical values, reducing the number of bits required exponentially. More on how this works is explained in section 2.3 (p. 24).

These n qubits can all be manipulated simultaneously using *quantum parallelism*, a method that applies quantum entanglement. Quantum parallelism performs a quantum operation on a superposition of inputs to produce a superposition of outputs. The drawback of this is that once an observation is made and the state collapses, the superposition is lost. A small example of this would be to apply the X gate on $|q\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ (the superposed input), which results in the X gate being applied on both a state $|0\rangle$ and $|1\rangle$ simultaneously to produce:

$$X|q\rangle = \frac{1}{\sqrt{2}}(|1\rangle - |0\rangle), \tag{1.14}$$

which is a superposed output.

Quantum interference can be used to reinforce the probability of obtaining the desired result and reduce or even annihilate the unwanted result via *constructive* or *destructive* interference, respectively. More on the applications of superposition, entanglement, and interference can be found in detail in Brassard et al. (1998).

However, since even the cutting-edge quantum computers are still in try-out phases, most potential gains (and losses) of applying quantum computing are not yet proven practically. Smaller circuits can run properly but, once the circuits rake up the amount of qubits, execution on a quantum computing becomes more difficult. To put it into perspective, the timeline of the capabilities in quantum computing is often described in three stages: Infancy, Noisy Intermediate Scale Quantum (NISQ), and Fault Tolerant (FT). The infancy stage is where only some quantum operations can be executed on qubits, but no real circuits can be executed entirely. The NISQ stage is where quantum computing is right now; small circuits can be executed entirely, but are often dominated by the noise that interferes with the result. Finally, there is the FT stage, where large quantum circuits can be executed and errors can be corrected to a stage where the results are no longer influenced significantly by them.

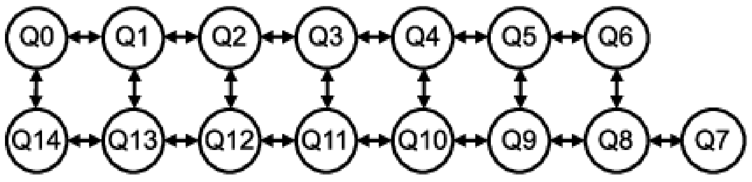


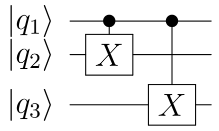
Figure 1.5: Schematic IBMQ Melbourne architectural structure applied in the Melbourne quantum computer of IBM. The qubits Qi are located on the nodes indicated by circles. (L. Liu and Dou 2020)

The noise that has been mentioned in the NISQ era is the main drawback in quantum computing as of now. The noise of a quantum computer is called *quantum decoherence*. In a nutshell, this means that each qubit in a quantum computer has a probability of losing information to the environment over time. The more qubits are in a system, the higher the probability that one qubit has lost too much information, causing the entire circuit to become useless. Therefore, large quantum circuits (constructed by many qubits) are more susceptible to noise.

In more detail, the structure of a quantum computer is a collection of qubits in a topological network. The goal that quantum computer hardware designers are mainly struggling with is to find a topological network where many qubits can be connected to each other with minimum noise. An example of the structure of a quantum computer is that of IBMQ Melbourne (L. Liu and Dou 2020), shown in figure 1.5. The graph created from this topology has vertices of degree-1, -2 or -3. This degree describes the number of neighboring qubits that one qubit has a direct connection with. The number of qubits required to execute a quantum circuit depends on the complexity of the circuit. For example, defining a quantum circuit with only Hadamard gate operations lets Qiskit pick three qubits from the topological structure to execute this circuit:

$$\begin{array}{l} |q_1\rangle \ -\boxed{H}- \\ |q_2\rangle \ -\boxed{H}- \\ |q_3\rangle \ -\boxed{H}- \end{array} \tag{1.15}$$

However, the choice of qubits is different if the quantum circuit contains entangled qubits via controlled gates, for example:



(1.16)

This circuit has also three qubits, with q_1 acting as the control qubit in two cX-gates indicated by the black dot and the X operation is indicated by the boxed X on q_2, q_3. For this circuit to be executed, all three qubits need a direct connection with each other. The topology of figure 1.5 (p. 14) does not support this connection initially, hence a qubit must be sacrificed to act as a *communicator* qubit. This is a qubit that passes along information to the other qubits. For example, in figure 1.5, $Q0, Q2, Q13$ can have a direct connection if $Q1$ acts as the communicator. That does make the circuit more susceptible to noise.

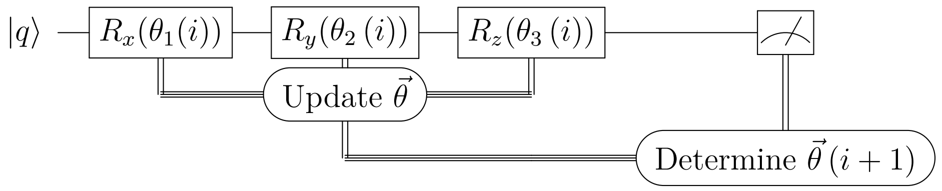
Regarding this noise, a different quantum computer is theorized. Although not necessarily relevant for this thesis, it is nevertheless an interesting concept. The proposition is a *topological quantum computer* (Freedman et al. 2003), which describes the quantum logic gates using the rotational motion between quasiparticles named *anyons* (Burton 2016). That being the case, such a quantum computer is still theoretical, since anyons have only been observed under very strict conditions (Nakamura et al. 2020), and such manipulations concerning anyons are still far away.

1.3.3 Hybrid and pure quantum algorithms

Using the knowledge from the previous section, a quantum circuit can be constructed. There are two methods of implementing a quantum algorithm on a quantum computer: a *hybrid* and a *pure* quantum model.

The former, a variational hybrid quantum-classical algorithm (hybrid) (McClean et al. 2016), uses the probabilistic nature of a quantum computer to its advantage for probabilistic calculations and uses the advantages of a classical computer to perform the many calculations and memory often required in machine learning.

To explain the structure of a basic hybrid model, the rotation gates from (1.12) are used to construct a *parametrized circuit*, e.g.



(1.17)

This is a circuit of one qubit q, which is manipulated by a R_x, R_y and R_z gate indicated by the boxes. These gates accept $\vec{\theta} = (\theta_1, \theta_2, \theta_3)$ as a parameter to create a state. The state is measured, indicated by the boxed meter, and causes the probabilistic state of $|q\rangle$ to collapse into either a $|1\rangle$ or a $|0\rangle$ with probability $|\alpha|^2$ or $|\beta|^2$ according to (1.5), respectively.[3] This result is passed on classically (indicated by the double line) to a classical machine to determine a new vector $\vec{\theta}\,(i+1)$, which can be applied to the R-gates for the next iteration. The user can define a desired state $|q_{\text{final}}\rangle$ according to (1.5) and pass an initial $\vec{\theta}_{\text{init}}$ to this circuit to create the first quantum state $|q_1\rangle$. $\vec{\theta}\,(i+1)$ can for example be determined using a loss function between $|q_1\rangle$ and $|q_{\text{final}}\rangle$.

This method can be used to create a hybrid convolutional neural network like in J. Liu et al. (2019), where quantum computing is used for the feature mapping process during the creation of a neural network, which is computationally the hardest part for a classical computer.

A *pure* quantum circuit uses the quantum computer to its fullest potential by performing everything on a quantum computer. This has its setbacks. For example, a quantum computer does not have a proper Random Acces Memory (RAM), so loading data onto the quantum computer provides a challenge.

Furthermore, the quantum circuit used to perform the operations has a certain *depth* $\mathcal{D}$. Think of it as the number of lines that is in a script that needs to be executed by the CPU: some lines can be run in parallel, but some lines must wait for other lines before they can be executed. For example, take the circuit from (1.15). Each quantum gate can be executed separately without being influenced by the other qubits. This means that everything can be performed in one action, making the depth of the circuit $\mathcal{D} = 1$. In the circuit from (1.16), the two controlled operations cannot be executed simultaneously and must be executed in order of occurrence. This makes the depth of that circuit $\mathcal{D} = 2$. The depth of the circuit in (1.17) would be $\mathcal{D} = 3$, since each rotation gate must be executed in the specified order and cannot be executed in parallel.

In addition, it is also very hard to compare a quantum and classical method against each other, since it is unclear if both methods are the best out there. Furthermore, by trying to replicate a classical method on a quantum computer one can be limited by the classical methods and forget to apply the benefits available in quantum computing, e.g. the use of complex data.

1.3.4 Translating the kNN to a quantum algorithm

The goal is now, with this knowledge, to describe the kNN into a pure quantum algorithm. This is done by going through the kNN algorithm step-by-step according to section 1.2 and describe the methods to translate these to a quantum algorithm.

[3]One measurement results in either a 1 or a 0 and only multiple measurements can approach the amplitudes α, β of $|q\rangle$.

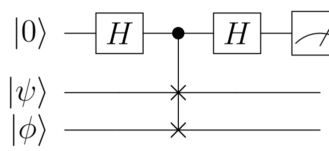
Determining distance

Instead of translating a classical distance measurement into a quantum circuit, one can use the nature of quantum states to determine the *fidelity* between two quantum states, say ψ and ϕ, which represents the overlap that these two states have. This can act as a distance measurement of some sort. The fidelity of these states is then given by $\|\langle\psi|\phi\rangle\|^2$, with $\langle\psi|\phi\rangle$ as:

$$\langle\psi|\phi\rangle = |\psi\rangle^\dagger \cdot |\phi\rangle, \tag{1.18}$$

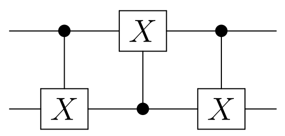
where $|\psi\rangle, |\psi\rangle$ are defined according to (1.5). Mathematically, this represents the dot product between two vectors and is directly relatable to the cosine similarity (1.1).

This equation can be translated into a quantum circuit by following the method from Buhrman et al. (2001):



(1.19)

The qubits $|\phi\rangle$ and $|\psi\rangle$ represent the states of which the fidelity needs to be measured. The upper qubit acts as the control qubit and is initialised in state $|0\rangle$. This method of measuring the fidelity is fittingly called the "$SWAP$Test-method", since it fundamentally works by the controlled $SWAP$ ($cSWAP$ or Fredkin) gate (Fredkin and Toffoli 1982). This is a combination of the $SWAP$ gate (1.8) and the controlled operation (1.11). An important thing to note here is that the $SWAP$-gate is not a fundamental gate but a combination of multiple fundamental gates (the cX gates):



(1.20)

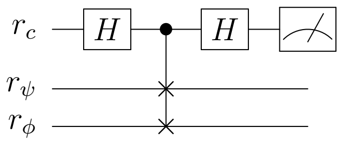
This means that its depth is $\mathcal{D} = 3$, which higher than what (1.23) seems to show (which is 1).

The measurement on the circuit in (1.23) is done on the upper qubit, and is indicated by the box with the measurement icon in it. The probability of measuring the qubit in states $|0\rangle$ or $|1\rangle$ is determined by Afham et al. (2020) as:

$$\mathbb{P}(0) = \frac{1}{2} + \frac{1}{2}\|\langle\psi|\phi\rangle\|^2, \quad \mathbb{P}(1) = \frac{1}{2} - \frac{1}{2}\|\langle\psi|\phi\rangle\|^2, \tag{1.21}$$

which now contains the fidelity values that can be retrieved via $\mathbb{P}(0) - \mathbb{P}(1)$. For clarity: by executing the quantum circuit once, the output will be just a 0 or a 1, and (1.21) must be approached by executing the circuit multiple times.

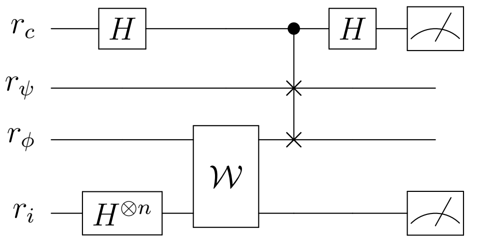
If the states $|\psi\rangle, |\phi\rangle$ are described by multiple qubits, say: $|\phi\rangle = |\phi_1\phi_2 \cdots \phi_n\rangle$, these can be separated into n qubits and the *cSWAP* is expanded to n *cSWAP* operations, e.g. $SWAP|\psi_i\rangle|\phi_i\rangle$. To prevent having to write each state in all n qubits $|\psi_i\rangle, |\phi_i\rangle$ the states will be described as *registers* r_ψ, r_ϕ consisting of multiple qubits describing one state:



(1.22)

Please note that the control qubit $|0\rangle$ is also written as a register r_c. This is for consistency, and is a register consisting of only one qubit.

This circuit for fidelity measurement has, however, a drawback: it measures only the overlap between two states, in this case ψ and ϕ. But what if there is a database with N data points, all described by $\Phi = \{\phi_1, \ldots, \phi_N\}$? For that, a modified circuit is used from Afham et al. (2020):



(1.23)

Here, the r_c identifies the control register again, which is in this case always one qubit. r_ψ, r_ϕ describe the registers for the state ψ to measure the fidelity, and the training vector ϕ over which to measure the fidelity.

The addition in comparison to the circuit in (1.22) is the r_i register, an oracle $\mathcal{W}$, and a measurement on r_i. The register r_i represents the *computational basis*. This is a register with n qubits, where $n = \log_2 N$, with N the number of data points in the training data set. All n qubits are brought in superposition via the $H^{\otimes n}$ gate creating the state $|i\rangle$ with $i \in [1, N]$. In detail, the computational basis is described by $|i\rangle = |\vec{d}\rangle$, where $\vec{d} = d_0 d_1 \cdots d_n \in \{0,1\}^n$ is a binary representation of the ith data point in Φ.

The state $|i\rangle$ is used as an index to identify which state from Φ needs to be applied to r_ϕ with the help of the *Oracle* $\mathcal{W}$:

$$\mathcal{W}|i\rangle|0\rangle = |i\rangle|\phi_i\rangle. \tag{1.24}$$

An Oracle is an operation that has some unknown property to get a desired result, in this case defined by equation (1.24). The goal is to find out what this Oracle is and how it operates to perform the action described in (1.24).

This circuit applies the method of quantum parallelism by bringing r_i into superposition of all states $|i\rangle$ at once, causing the Oracle $\mathcal{W}$ to initialize r_ϕ into a superposition of all data points $\phi_i \in \Phi$ with equation (1.24). Hence, the overlap between ψ and all states ϕ_i is measured simultaneously.

At the time of writing, the only applied distance measurement used on quantum computers has been the $SWAP$Test. There are some adaptations on it, like Lloyd, Mohseni, et al. (2013), but at its core it is still a measurement of the overlap of two vectors.

Find k closest

The circuit shown in equation 1.23 provides a solid method of simultaneous fidelity measurement of one state ψ against a whole set of states Φ. As shown in this circuit, a measurement has been added to the r_i register, which will be used to determine the values which will identify which state has the most overlap with ψ (i.e. which data point is closest). Afham et al. (2020) have worked out that the probability densities of the r_c register can be defined by:

$$p(0) = \frac{1}{2} + \frac{1}{2N}\sum_{i=1}^{N}\|\langle\psi|\phi\rangle\|^2, \quad p(1) = \frac{1}{2} - \frac{1}{2N}\sum_{i=1}^{N}\|\langle\psi|\phi\rangle\|^2, \tag{1.25}$$

and that the probability density of measuring the i-th outcome in register r_i is described by:

$$p_0(i) = \frac{1 + \|\langle\phi_i|\psi\rangle\|^2}{N + \sum_{j=1}^{N}\|\langle\phi_j|\psi\rangle\|^2}, \quad p_1(i) = \frac{1 - \|\langle\phi_i|\psi\rangle\|^2}{N - \sum_{j=1}^{N}\|\langle\phi_j|\psi\rangle\|^2}. \tag{1.26}$$

They define a new variable named the *contrast* $q(i)$ which is the difference between the probabilities $p_0(i)$ and $p_1(i)$ for the i-th outcome:

$$\begin{aligned} q(i) &= p_0(i) - p_1(i) \\ &= \frac{1 + F_i}{N + \sum_{j=1}^{N} F_j} - \frac{1 - F_i}{N - \sum_{j=1}^{N} F_j} \\ &= \frac{2\left(F_i - \langle F\rangle\right)}{N\left(1 - \langle F\rangle^2\right)}, \end{aligned} \tag{1.27}$$

where $F_i = \|\langle\psi|\phi_i\rangle\|^2$ is the fidelity and $\langle F\rangle = \sum_{j=1}^{N} F_j/N$ is the average fidelity of $|\psi\rangle$ with all the states $\{|\phi_i\rangle\}$.

This contrast is now directly proportional to the fidelity, and Afham et al. (2020) uses this for the kNN classification. The reason for not rewriting this to a direct fidelity measurement remains unclear and, since this deemed more logical, the equations have been manipulated in such a way that the fidelity can be found via:

$$F_i = \frac{N}{2} q(i) \left(1 - R^2\right) + R \tag{1.28}$$

where $R = p(0) - p(1)$.

These values cannot be received directly from running the circuit once. The circuit needs to be executed a total number of T times, where each measurement is counted, e.g. $r_c = 0 : T_0, r_c = 1 : T_1$ and $r_i = i$ given $r_c = 0 : c_0(i), r_i = i$ given $r_c = 1 : c_1(i)$. This means that $T_0 = \sum_i c_o(i)$ and $T_1 = \sum_i c_1(i)$. Using these values, the probability densities can be estimated via:

$$\begin{aligned} p(n) &\sim \bar{p}(n) = T_n/T, \\ p_n(i) &\sim \bar{p}_n(i) = c_n(i)/T_n, \quad ` \\ q(i) &\sim \bar{q}(i) = \bar{p}_0(i) - \bar{p}_1(i). \end{aligned} \tag{1.29}$$

Majority vote

The majority voting is a matter of defining the distance D as $1 - F_i$, sorting the list, and obtaining the most occurring class in the k number of data points with the lowest D.

Chapter 2

Method

This chapter describes the approach to implementing a QkNN using Qiskit in Python. It starts with section 2.1, which handles the probable reuse and accessibility cases for the code, which will then be described more in-depth in the rest of the chapter. Section 2.2 heads into the datasets used in this project to generate the results and benchmark tests, which need to be encoded according to 2.3. Finally, the details on the construction of the QkNN, and how Afham et al. (2020) is used to create the program and how the kNNs are initialized, are described in sections 2.4 and 2.5.

2.1 Reuse and accessibility

Studying and reusing someone else's code is often avoided due to it being conceived as hard and time-consuming.[1] To prevent this, one of the goals of this thesis is to make the provided code as accessible as possible given the current (quantum) programming knowledge, such that it is inviting others to further develop the current code. The code has been published on PyPI for people to access and install on their system for (Kok 2020a). To help with understanding the code, a full documentation page is brought to life, hosted by Read the Docs (Kok 2020b), and a GitHub page exists for any voluntary contribution (Kok 2020c). The documentation also includes a quick-start located at the welcome page to guide with the understanding of the process.

[1]It is even so bad, that people often rather rewrite it in their code, rather than trying to decipher the existing code.

2.2 Data

Three sets of data are used for this project: firstly, the Iris data set from Anderson (1936), which is provided within `scikit-learn`, is used as a simple benchmark for how the algorithm performs on easily accessible data.

Secondly, the dataset from Hofmann (1994) (hereafter referred to as “the German credit data”) is used, which describes credit data from multiple account holders from Germany, and is mostly applicable in the financial field. Finally, a HEP dataset is used (DarkMachines 2020), which is an unsupervised data set describing multiple types of collision events in a Large Hadron Collider (LHC).

These three data sets are real-valued since the application of this QkNN is to test its practical use. A complex data set can be beneficial for the QkNN, using a unique process of a quantum computer to its advantage possibly resulting in a speedup. However, since this currently does not represent real-world data it will be left out of this project.

2.2.1 Pre-analysis

Since the number of qubits in the quantum circuit are scarce, the resources for describing the data is limited. The way the quantum circuit works right now, one circuit is made per test data point, containing the entire training data. Hence, for each data point that needs to be classified, a new quantum circuit is made. This means that the number of test data is not the largest limiting factor, since the quantum circuit can be recreated once the machine runs out of memory. The limiting factor is the dimensionality D and the number of data N used for the training method: the runtime remains acceptable with these values at 8 and 16 at maximum for both.[2] Values not following the relation of 2^n can be chosen but can be considered a waste of qubit resources. More on this will be explained in 2.3.

Looking at the dimensionality issue it seems that this is only a problem for the German credit and HEP data set, since the Iris data set has a dimensionality of 4. Since it is out of the scope of this project to look for a relation in data in the German data, the first n parameters are chosen to contribute to the classification. Since the theoretical background is more physics-based, a (very) rough numerical data set can be created to represent the HEP data.

DarkMachines data (HEP data)

One of the goals at CERN is to use the LHC to provide experimental results in the research on the subject of Beyond the Standard Model (BSM) physics (Lykken 2011). BSM physics is a theory to explain the shortcomings of the Standard Model

[2]The machine used for this has a 6-core processor at 2,8GHz-4GHz with 16GB DDR4 RAM.

in physics and a big research subject within BSM is that of Super Symmetry (SUSY). The SUSY theory predicts a symmetric counter particle for each particle in the Standard Model, and the model created to accompany this theory is the Minimal Supersymmetric Standard Model (MSSM) (Haber and Kane 1985). The experimental results exist of collisions between accelerated protons (*pp* collisions) to investigate the proton's structure. The resulting data are either direct or indirect observations of elementary particles. If this theory is correct, the collision events are expected to be observed within the LHC.

The data delivered by DarkMachines (2020) are simulation data of the LHC describing possible observations of possible events according to the MSSM.

The data is delivered in a `.csv` file as follows:

```
event ID; process ID; event weight; MET; METphi; obj1, E1, pt1↲
eta1, phi1; obj2, E2, pt2, eta2, phi2; ...
```

The parameters mean the following:

- `event ID`: this is the ID of the event. By no means is this collision data, but just the number of a generated event. No training data;
- `process ID`: this describes the event that took place, and which the algorithm must identify. The possible processes are e.g. `ttbar` or `4top`;
- `event weight`: integrated event weight of the simulated collision, which tells the probability that certain events are produced (as given by the simulator);
- `MET; METphi`: magnitude and azimuthal angle of the missing transverse energy vector of the event, respectively;
- `obj`: the produced object (or particle) from the event, which can be either `j`, `b`, `m+/m-` `e+/e-` or `g`, which are a jet, b-jet, muon, electron or photon, respectively and the muon and electron can also hold a charge value indicated by the `+` or `-`;
- `E, pt, eta, phi`: the four-vector of the specified object, in order the energy, transverse component of the momentum, the θ and ϕ angles.

From this data set, a channel is selected (channel 1) containing SUSY gluino-gluino production (supersymmetric partner of the gluon) with an 800 GeV neutralino (supersymmetric partner of the gluon), where gluinos decay to jets (`b/j`) and missing energy (`MET`). (Mariotto and Rodriguez 2008) The goal of the classifier is to distinguish these events from background observations, also provided by this data set. This data has 2 dimensions dedicated to the `MET` and `METPhi`, followed by a combination of objects (`obj`) with their four vectors `E, pt, eta, phi`. The number of objects per event varies and can at maximum be 16. This means that

16 four-vectors deliver a 64-dimensional data point, which is not suited for the quantum hardware. The goal is to reduce this data to a dimensionality suitable for the quantum circuit, so around 8 dimensions. Two of these parameters, the `MET` and `METPhi`, are continuous values. This means that these two can be used as two parameters without any manipulation.

The rest of the information must be condensed into the remaining 6 dimensions. A lot of information can already be obtained by following the Feynman rules of the MSSM (Rosiek 1990), which state that certain outputs from a gluino-gluino diagram are physically not possible. If an output is observed which is physically not possible, one could conclude that the gluino-gluino diagram event did not occur. Hence, instead of using all the data provided, the data is reduced to an 8-dimensional set of "meta-features" containing:

`MET; METphi; n_j; n_b; n_m; n_e; n_g; n_0`,

which are the missing energies (untouched), and the number of times a certain object is observed, including the number of times there is an empty observation. The empty observation work as follows: the number of observed objects vary from 1 to 16, meaning that when e.g. 1 jet is observed, `n_j` is 1, and `n_0` is 15.

German credit and Iris data

The Iris dataset (Anderson 1936, Fisher 1936) is 4-dimensional and contains three classes. It consists of length and width measurements of sepals and petals from three different flowers: the Iris Setosa, Iris Versicolour, and Iris Virginica. In short, the dimensionality need not be changed, and only two of the three classes will be used to fit the comparison with the other two, binary classified, datasets.

The German credit data, however, has 25 dimensions. It comes with two files: the original and a numeric one provided by Bartley (2016). It is a dataset containing information about checking accounts, their owners (including personal info like sex and age), their housing and employment status. Bartley has found a method to describe these labels in all integer or binary data for numeric algorithms. Thus, when varying the number of dimensions, the variables will be picked in order of the data set as provided.

2.3 Encoding

All data provided to the quantum circuit is saved to the qubits in the circuit. This is done by *encoding* the data, a method that is also done on a classical computer by encoding data to binary. For example: if one were to encode a number to a classical computer it is translated to a chain of 0s and 1s: $\vec{d} = d_0 d_1 \cdots d_n \in \{0,1\}^n$. $\vec{d}$ describes a sum, which is calculated via:

$$\begin{pmatrix} d_0 & d_1 & \cdots & d_n \end{pmatrix} \cdot \begin{pmatrix} 2^0 \\ 2^1 \\ \vdots \\ 2^n \end{pmatrix}, \tag{2.1}$$

and can create any real number, e.g. $6 = 110$ or $9 = 1001$. This same method can be applied on a quantum computer.

2.3.1 Digital encoding

Digital encoding, also known as *binary encoding*, is a method to describe data in a binary string of 1s and 0s just as on a classical computer. This thesis does not utilize binary encoding but an understanding of the concept is crucial. Let N and $\boldsymbol{d}_j = \{d_j^{(k)}\}_{k=1}^m$ $(d_j^{(k)} = 0, 1, j = 1, \cdots, N)$ be the number of binary data provided and the binary bitstrings. The equation for digital encoding can be described by:

$$\frac{1}{\sqrt{N}} \sum_{j=1}^{N} |j\rangle |\boldsymbol{d}_j\rangle. \tag{2.2}$$

The binary string that results can now be assigned to the same number of qubits which simply hold a 0 or 1. The distance can then be easily determined via Hamming distance calculation. This way of encoding is very basic and can in principle describe every state. The disadvantage is that it needs many qubits to do so, which is at this time and date not yet feasible. It also does not take advantage of the characteristic superpositions of qubits whilst describing data.

2.3.2 Analog encoding

Another method to encode data is *analog* encoding, also known as *amplitude encoding*. It is the method of describing variable data within the amplitudes of qubits, and has already been briefly mentioned in (1.13) (p. 13). The values of a vector of length N can be described in a quantum state by normalizing the vector and assigning each value to each amplitude of the register. For example, two vectors $\vec{a} = (2, 3)$ and $\vec{b} = (2, 3, 1, 4)$ could be described by:

$$\begin{aligned} |q_a\rangle &= \alpha_1|0\rangle + \alpha_2|1\rangle \\ &= 2|0\rangle + 3|1\rangle; \\ |q_b\rangle &= \beta_1|00\rangle + \beta_2|01\rangle + \beta_3|10\rangle + \beta_4|11\rangle \\ &= 2|00\rangle + 3|01\rangle + 1|10\rangle + 4|11\rangle, \end{aligned} \tag{2.3}$$

where $|q_a\rangle$ is one qubit describing $\vec{a}$ in the amplitudes $\alpha_i \in \mathbb{C}$ and $|q_b\rangle$ are two qubits describing $\vec{b}$ in the amplitudes $\beta_i \in \mathbb{C}$. The quantum states $|0\rangle, |1\rangle$ and $|00\rangle, \ldots, |11\rangle$ are the basis states of $|q_a\rangle$ and $|q_b\rangle$ according to (1.3) and (1.7), respectively. This also shows that a vector with $D = 4$ needs two qubits to be encoded on a quantum computer. More general, a vector with dimension D needs $d = log_2(d)$ qubits to be encoded. If a vector is not a power of two, i.e. $D \neq 2^d$, the vector can be filled with a number of 0's. This is not a problem for the encoding other than that it can be considered a waste of qubit resources.

However, since the qubits are brought to a quantum state, there is an important property to keep in mind. A quantum state needs to object to the properties of Hilbert space, meaning that

$$\langle q|q\rangle = 1. \tag{2.4}$$

This means that the vector $\vec{a}$ needs to be normalised before the values of the vector can be encoded into the amplitudes. The following equation describes the encoding for an arbitrary state:

$$\sum_{j=1}^{N} c_j|j\rangle, \tag{2.5}$$

where $\{c_j\}_{j=1}^{N}$ is normalised to statisfy $\sum_{j=1}^{N} |c_j|^2 = 1$, so that the property in (2.4) is met. Using the example of (2.3), the encoded states would actually look like:

$$\begin{aligned}
|q_a\rangle &= \frac{1}{A}\Big(2|0\rangle + 3|1\rangle\Big) \\
&= \frac{1}{\sqrt{2^2+3^2}}\Big(2|0\rangle + 3|1\rangle\Big); \\
|q_b\rangle &= \frac{1}{B}\Big(2|00\rangle + 3|01\rangle + 1|10\rangle + 4|11\rangle\Big) \\
&= \frac{1}{\sqrt{2^2+3^2+1^2+4^2}}\Big(2|00\rangle + 3|01\rangle + 1|10\rangle + 4|11\rangle\Big)
\end{aligned} \tag{2.6}$$

This method is incorporated in the QkNN in Python by using the `einsum` method provided by Numpy, which performs the Einstein summation:

```python
def encode(classical_data):
    # sum up every row of the matrix to get the lengths for each
    #  row via a_ij * a_ij = A_i
    amplitudes = np.sqrt(np.einsum('ij,ij->i', classical_data,
    classical_data))

    # set zero amplitudes to 1 to prevent division through zero
    amplitudes[amplitudes == 0] = 1

    # normalise the data by dividing the original through
    #  the amplitude
    normalised_data = classical_data / amplitudes[:, np.newaxis]

    return normalised_data
```

2.4 Implementing the QkNN

Now that the data has been prepared, the QkNN can be set up to start classifying it. As mentioned in 1.3, Qiskit will be the SDK of use throughout this project, and the circuits will be simulated using the `qasm_simulator`. This simulator will run the circuit as if it is run on a quantum computer, meaning that it will have to run multiple times to approach the state vector produced by the quantum circuit. An alert reader might propose to instead use the `statevector_simulator`, since this will just give the state vector as a direct result. However, this seems a bit unfair since it does not approach a “real” physical result (i.e. “directly observable”).

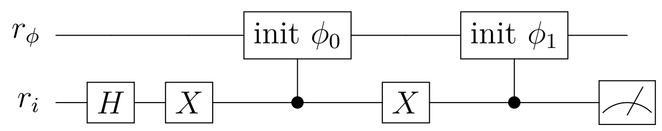
First, the quantum circuit as shown in (1.23) has to be created within Qiskit. The gates are applied in a fashion as shown in section 1.3, but the Oracle needs some more attention, which will be given in the section below.

2.4.1 Oracle

As mentioned before in section 1.3.4 (p. 16), an Oracle is an operation that has some unknown property to get a desired result. The goal is to find out what this Oracle is and how it operates to get the desired result. The Oracle is only briefly touched upon in Afham et al. (2020), so in order to produce the desired result as shown in (1.24), an algorithm needed to be designed from scratch. To repeat the goal of the Oracle $\mathcal{W}$ described in (1.24): it needs to initialise a register r_ϕ into a state describing a data point $\phi_i \in \Phi$ if a register r_i is in state $|i\rangle$:

$$\mathcal{W}|i\rangle|0\rangle = |i\rangle|\phi_i\rangle.$$

The following idea is implemented: create a combination of X-gates to set r_i in all possible combinations of $1^{\otimes n}$, and then check the configuration via a controlled gate. For a register with one qubit it would look as follows:



(2.7)

So, should r_i be in state $|i\rangle = |0\rangle$, the algorithm applies the first X-gate, bringing r_i to $|1\rangle$, making the first controlled operation possible. If r_i was in state $|1\rangle$ however, it would pass the first controlled operation and be brought back to state $|1\rangle$ after the second X-gate, making sure that ϕ_2 is applied in the second controlled operation. This can be optimized to contain only one X-gate to decrease circuit depth, but it is written like this for better readability for now.

To write this for an arbitrary length for r_i, the location where to apply the X-gates can be determined by the expression $|i\rangle XOR|i-1\rangle \otimes X^{\otimes n}$ e.g. for $n = 8$, if one wants to know how to check the fifth state:

$$\begin{aligned} |5\rangle XOR|4\rangle &= |1010\rangle XOR|0010\rangle \\ &= |1101\rangle, \end{aligned} \tag{2.8}$$

meaning the X-gates need to be applied to all qubits except the third one.

This is described in the code in a method called `where_to_apply_x()`, which accepts the length of the binary number as an integer, and returns a list of all the combinations where the X-gates must be applied to get all possible $|i\rangle$ states:

```python
def where_to_apply_x(bin_number_length: int) -> List:
    """ Create an array to apply X-gates systematically to create
    all possible register combinations.

    This method returns the indices on where to apply X-gates on a
    quantum register with n qubits to generate all possible binary
    numbers on that register.
    """
    powers_of_two = 2 ** np.arange(bin_number_length)
    indices = \
        [
            [
                ind for ind, v in enumerate(powers_of_two)
                if v & (pos ^ (pos - 1)) == v
            ] for pos in range(2 ** bin_number_length)
        ]
    return indices
```

Since the computational basis r_i is brought into a superposition of all states $|i\rangle$, the register r_ϕ is brought into a superposition of all training states $\phi_i \in \Phi$ because of this Oracle $\mathcal{W}$.

2.5 Setting up the kNNs

The kNN classifier that is used in this thesis is the `KNeighborsClassifier` provided by `scikit-learn`. This class has some base algorithms implemented to use the kNN to its fullest potential, but since these are not implemented in the QkNN, these need to be disabled.

First, the cosine similarity (1.1) must be described as a method to pass to the kNN:

```python
def cos_sim(x, y):
    """Cosine similarity."""
    return x @ y / (np.sqrt(x @ x) * np.sqrt(y @ y))


def cos_dist(x, y):
    """Distance value using the cosine similarity."""
    return 1 - cos_sim(x, y) ** 2
```

These methods can now be passed on to the kNN.

The method `cos_dist` is used to initialize the kNN in the code as follows:

```python
def setup_knn():
    """
    Initialise an empty KNeighborsClassifier with brute parameters
    and own defined squared cosine similarity distance measurement.
    :return: KNeighborsClassifier
    """
    # intialising the knn model
    model = neighbors.KNeighborsClassifier(
        n_neighbors=3,
        algorithm='brute',
        # brute-force computation for NNs, needed if using cos_dist
        n_jobs=None,  # obviously not implemented in the qknn
        metric=cos_dist,  # self-defined cos_dist calculation above
    )

    return model
```

The QkNN is designed to be similar to the `KNeighborsClassifier`, but needs some extra setup for the inclusion of the quantum backend and instance:

```python
def setup_qknn():
    """
    Initialises an empty QKNeighborsClassifier with backend and
    instance
    """
    # initialising the quantum instance
    backend = qk.BasicAer.get_backend('qasm_simulator')
    instance = aqua.QuantumInstance(backend, shots=20000)

    # initialising the qknn model
    model = qknc.QKNeighborsClassifier(
        n_neighbors=3,
        quantum_instance=instance
    )

    return model
```

These models are then used for testing and performing benchmarks.

Chapter 3

Results

This chapter shows the results of the QkNN and the kNN. It starts with the results of the encoding in section 3.1. Afterward, the QkNN and kNN classifiers are benchmarked in section 3.2, which is followed by a more in-depth result of the QkNN and its variance in section 3.3. Finally, an indication of the runtime is shown in section 3.4.

3.1 Encoding

The effect of encoding the dataset as explained in 2.3 is shown in figure 3.1 (p. 32). The figure shows two parameters from the Iris dataset and the effect of standardizing (or normalizing using the Z-score method) and encoding the data. If the dataset had not been standardized first, the rightmost figure would only have one quadrant of the circle filled with all data points, meaning a much smaller window for classification.

This figure shows that the classification can be visualized as a compass: the hand of the compass will point in the direction of the unclassified data point and the k closest training data points to this unclassified data point determine the class.[1]

3.2 Accuracy benchmarks

The results of the benchmark between the kNN and QkNN are shown in figure 3.2. The figure shows the accuracy values that the kNN and the QkNN have predicted for 16 data points from all three data sets by using 16 values as "training" data.

[1]To complete the figure of speech, the clusters of data representing one of the three classes can be seen as the "cardinal directions". In this case there would be three classes, hence three cardinal directions.

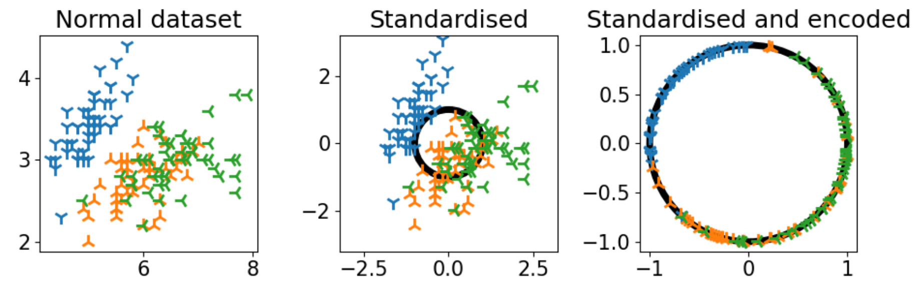


Figure 3.1: Visualisation of the encoding method on the sepal length and sepal width on all three classes (blue, orange and green) of the Iris data. From left to right: the untouched dataset as provided from Anderson (1936), the same dataset standardized using `scikit-learn`, and the standardized (or Z-score normalized) dataset encoded to be loaded onto the quantum circuit.

The data are binarily labeled (either 0 or 1), and the support for each label is 50/50 in both the training and the test data. The accuracies between the kNN and the QkNN vary from each other, except for $D = 16$ for the German credit data. This is despite the effort made to make the kNN identical in distance measurement and voting compared to the QkNN (at least on classical data). Furthermore, the QkNN outperforms the kNN in the case of the German credit data, but the kNN outperforms the QkNN in the other two data sets.

Figure 3.3 shows the influence on three basic preprocessing methods on the results of the kNN's. Each figure shows the accuracy values that the kNN and the QkNN have predicted for 16 data points from all three data sets by using 16 values as "training" data, similar to figure 3.2.

The four plots show different pre-processing methods of the data, which are from left to right: no preprocessing, standardizing, robust and minmax method. The three preprocessing methods are provided by `scikit-learn` and are usually chosen together with the kNN classifying method. Since the success of each preprocessing method is very dependent on the data and this project uses three very different data sets, no one preprocessing method can be used on all three data sets. Hence, this figure is made to show the influence of each method on the individual data sets. Each plot then shows the accuracy versus the number of parameters (or dimension D), where this is varied from 4, 8, 16. This variability is chosen to see the influence of more variables on the classification.

The first thing to observe is that the classical and quantum method vary from each other, and seem to follow different relations concerning the preprocessing method and number of parameters. Also, the classifiers had an accuracy lower

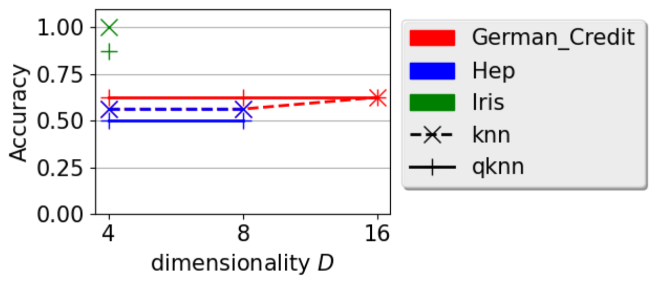


Figure 3.2: The accuracies over 16 predictions made by the QkNN (plus, solid) and the kNN (cross, dashed) with 16 training points of dimensionality 4, 8 and 16 and binary labels with 0.5 coverage each, for the German credit (red), the HEP (blue) and Iris dataset (green). The classifications for the data are distinguishing between: fraudulent from non-fraudulent bank users, 1.6 TeV gluino-gluino events from background and Iris setosa from Iris virginica in the German credit, HEP and Iris data respectively.

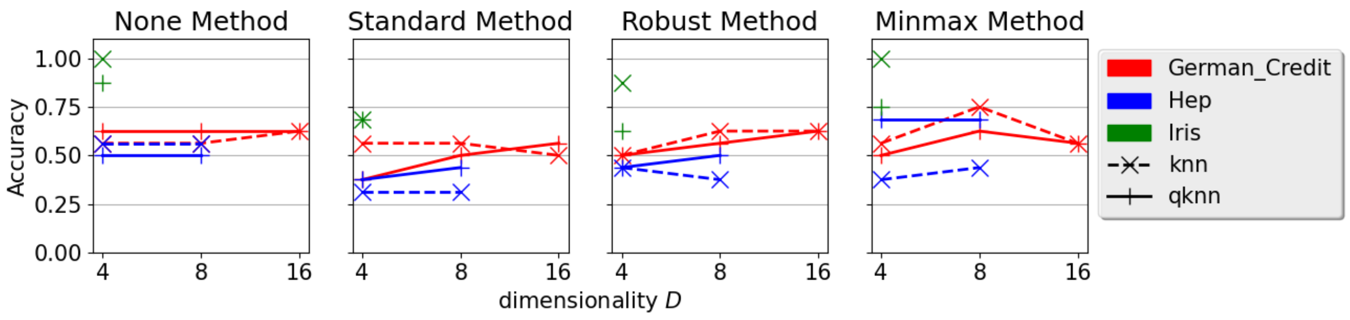


Figure 3.3: The accuracies over 16 predictions made by the QkNN (plus, solid) and the kNN (cross, dashed) with 16 training points of dimensionality 4, 8 and 16 and binary labels with 0.5 coverage each, for the German credit (red), the HEP (blue) and Iris dataset (green). From left to right: the data is preprocessed using no method, standardization (Z-score normalization), robust and min-max method. The classifications for the data are distinguishing between: fraudulent from non-fraudulent bank users, 1.6 TeV gluino-gluino events from background and Iris setosa from Iris virginica in the German credit, HEP and Iris data respectively.

than 0.5 on the HEP data when preprocessed.

Furthermore, from the figure can be seen that the Iris dataset works best without preprocessing for both classifiers. Notable as well is the 1.0 accuracy of the kNN with the minmax preprocessing. The HEP dataset and German credit data both perform best with 8 variables and using the minmax preprocessing.

That being the case, the QkNN did (barely) outperform the kNN on several occasions: the HEP data without preprocessing and the German data when standardized (and using 4 or 8 variables) and across the board when using the robust or minmax method (ignoring the ties).

3.3 Number of shots

The accuracy values with their standard deviation as a function of the number of shots made by the `qasm_simulator` are shown in figure 3.4. The number of shots is in order: 10, 100, 1.000, 10.000, 20.000, and 65.535, where the last one is the possible maximum number of shots for this simulator purely to test if setting the simulator to the maximum number of shots possible will affect the precision of the result. The parameters used to classify the data are based on the result shown in figure 3.3: the dimensionality is kept as low as possible for better runtimes, thus 4, 8, and 16 for the German credit, HEP and Iris data. Furthermore, only the HEP data is preprocessed by using the minmax method. Finally, to provide an idea of the variability on the predictions from the QkNN, each run is repeated 10 times.

To start, the German credit data shows no increase in consistency, except at 20.000 shots. However, since the consistency at 65.535 is back to the previous results, this may be caused by chance. Next in line, the HEP dataset shows a bit better consistency, since the outliers in accuracies are only considered outliers on the mean and quartiles of the data. Finally, the accuracy over the Iris dataset shows an increase per number of shots, and a slight increase in consistency as the shots increase.

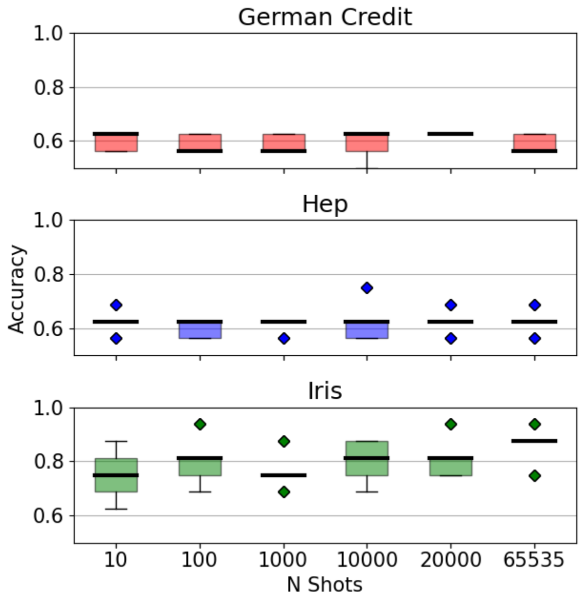


Figure 3.4: Boxplots of 10 repetitions of the same classification of 16 test data made by the QkNN for the three datasets (German credit in red, HEP in blue and Iris in green) as function of the number of shots made by the `qasm_simulator`. Each data point in this plot shows 10 accuracy values via a colored box with whiskers and flier points. The colored box extends from the lower to upper quartile values of the spread out accuracies, with a bold black line at the median. The whiskers extend from the box to show the range of the data and colored flier points (diamonds) are those past the end of the whiskers. The dimensionality of the data is 4 for Iris and German credit, and 8 for HEP, and only the HEP dataset is preprocessed by using the minmax method. The classifications for the data are distinguishing between: fraudulent from non-fraudulent bank users, 1.6 TeV gluino-gluino events from background and Iris setosa from Iris virginica in the German credit, HEP and Iris data respectively.

Table 3.1: The runtime in milliseconds of the QkNN and kNN based on 16, 16-dimensional data points from the German credit data. The runtimes are normalized per transpiled quantum circuit by dividing the total runtimes through 16.

Process	**Time (ms)**
Build time of the QkNN	8261
Simulation of the QkNN	17246
Total kNN runtime	0.3

3.4 Runtime

To determine the complexity and execution time of the QkNN, the creation and transpiling of the quantum circuit is timed by using a dataset with dimensionality $D = 16$, training size $M = 16$, and number of test vectors $N = 16$. The total time is then divided by 16 to have the times per quantum circuit. The results are shown in table 3.1, and show construction to simulation ratio of roughly 2:1. In comparison, the classical counterpart succeeds to do the entire classification within 5 milliseconds for all 16 data points.

A more in-depth result is shown in figure 3.5, which shows the influence of the complexity on the runtime of the kNN and QkNN for all three data sets. Note the logarithmic scale on the runtime as well, revealing that the runtime is not linearly, but exponentially scaling with complexity for the QkNN, and almost constant for the kNN. There are double results at $\log_2(ND) = 2^5$, since there were two runs performed with that complexity: one with $(N, D) = (8, 4)$, and one with $(N, D) = (16, 2)$. This differing result shows that N and D have an individual effect on the runtime.

Figure 3.6 shows the runtime of classifying a test vector using the QkNN. 16 test vectors from the German credit data are classified and averaged, and the depth is determined by transpiling the created QkNN quantum circuit using the rotational gates (1.12) and the cX-gate as basis gates. The runtimes T and depth $\mathcal{D}$ are on a base-10 logarithmic scale and shows an exponential relation. The German credit data is picked for runtime measurement since this has the highest reach in dimensionality. The other data sets show similar results at lower complexities, as can be seen in figure 3.5, but are not included to prevent clutter.

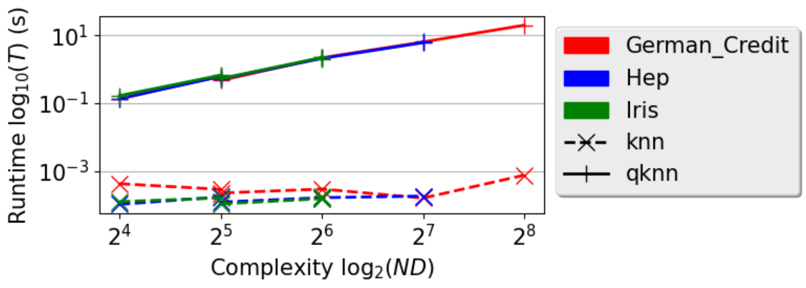


Figure 3.5: The runtimes (s) for classifying one test vector of the QkNN and kNN. The time is in $\log_{10}$ scale against the complexity $\log_2 ND$, with N the number of training points and D the dimensionality of the data, for a constant number of nearest neighbors $k = 3$. The runtime is determined from the start of the classification till the end by classifying 16 test vectors, and then divided by 16.

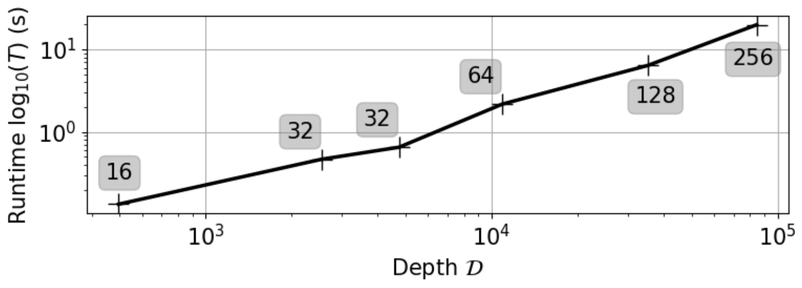


Figure 3.6: The average runtime (s) of classifying one test vector using the QkNN as function of the circuit depth $\log \mathcal{D}$. The boxes next to the crosses show the complexity ND. The circuit depth is determined by transpiling the circuit into the basis gates consisting of the rotation gates (1.12) and the cX-gate. The runtimes are average values determined by classifying 16 fraudulent from non-fraudulent bank users from the German credit data. There are two similar complexity values with different depths, which are two different combinations of (N, D): f.l.t.r. $(4, 8)$ and $(2, 16)$.

Chapter 4

Discussion

This thesis has researched the application of cutting-edge quantum computing algorithms by translating classically known methods to quantum algorithms. It shows the ease of creating a first quantum circuit and the development of quantum computing in general over the past years. The QkNN is a working algorithm that is capable of classifying data by measuring the fidelity between states and competes with the classical kNN modified to use the cosine similarity as distance measurement. This chapter provides a summation of the results gained and the methods used to obtain the results in similar order as the sections 4.1 and 4.2. In addition to this, ideas on how to follow up on these thoughts are provided in 4.3.

4.1 Method

Something important that may be critiqued on the QkNN is that it might be considered a hybrid instead of a pure quantum algorithm, since the sorting of the neighbors is done classically according to the method in Afham et al. (2020). Another method might be that of Blank et al. (2020), which have their labels assigned to their quantum register, such that the "sorting" is done in the quantum circuit itself, and that the probability distribution of the label observation describes the class of the unclassified data. It might be more complete to further implement this in `qiskit-quantum-knn`. The reason that it was not implemented in this work is that Blank et al. was released at the time of writing. The drawback that might be a result of this, is that the classification becomes a probability state. This can, however, also be considered a potential gain, because the QkNN would give an out-of-the-box uncertainty to the classification (although influenced by the noise of the quantum computer itself).

To continue on the QkNN, an adaptation mentioned in Lloyd, Mohseni, et al. (2013) is to have the distance calculation based on *centroid* distance calculation.

The kNN is transformed into a k-means method, where each class is defined by the centroid (or mean) of all data with that class. They define one vector pointing toward that centroid to load as a state on the quantum circuit and execute the same distance measurement to identify the new class. This can be beneficial since the QkNN would be more direct in its result; when the unit circle from figure 3.1 contains all data, it becomes too cluttered, like that example already shows. If this is reduced to vectors only representing the centroids of all classes, this would be much cleaner. An important thing to note, however, is that information is lost: what if a class contains two, spread-out clusters? The centroid would point to the middle, so classification would suffer from it. This does not mean that it is a useless adaptation to the QkNN, though. It just requires some extra thought to the process.

4.2 Results

Jumping from the method of the QkNN to its results, the first noticeable observation from the different predictions based on preprocessing method and dimensionality in figure 3.3, is that the QkNN and kNN actually have different accuracies. This is even after the effort of making the kNN and QkNN theoretically identical as described in 1.2 and 1.3.3. One of the explanations could be the variability of the quantum computer, as 3.4 shows.

Furthermore, the accuracies are quite poor. This can be caused by the following, looking back at the quote from Institute of Electrical and Engineers: the k-nearest neighbors method itself thrives under quantity of data, and this was not feasible with the resources available; if the quantity would be higher, the clusters that the kNN looks for to classify data are more prominent, and can “reach” further. Since the (Q)kNN has only worked with 16 training vectors, the prediction result suffers more than needed, and higher quality data is needed to compensate. Quantifying the quality of the data is a difficult task, and out of the scope of this project.

In the results shown in figure 3.4, something interesting can be observed as well. Both the German credit and HEP data seem to remain untouched by the number of shots, but the Iris data shows something expected. Its prediction and the consistency on it improve based on the number of shots. This means that, indeed, the variability in the predictions can be the influence of the probabilistic nature of the quantum computer, and that the German credit and HEP data are probably of too low quantity to show these results. If that is the case, the prediction accuracy can be improved by increasing the amount of shots that the quantum computer makes for measuring, provided the quantity of training data is sufficient.

Finally, the reason why the QkNN was considered in the first place, was that it would scale logarithmically with the number of training data and dimensionality

N, D. In an attempt to measure this, figure 3.5 shows the runtime based on $\log_2(ND)$ and figure 3.6 shows the runtime based on the circuit depth $\mathcal{D}$, which show an exponential relation for the QkNN. It is not a "real" result, however, since it is still based on a simulated quantum computer run locally on a classical machine, so the compilation and transpiling of the quantum circuit are probably negatively affected by this. The step from a NISQ computer toward a FT quantum computer would promise the possibility of larger data sets and higher complexity algorithms to be executed. This looks more promising in confirming the hypothetical runtime reduction. A temporary solution could be to transfer from the current SDK (Qiskit) to other available SDKs.

4.3 Follow-up

During this project, the booming research around quantum computing did not come to a stop: independently of this work, the authors of Park, Petruccione, et al. (2019a) attained the same algorithm/results for developing a solution to produce a QRAM. This parallel development motivates for further research and provides a great comparison between work.

On another topic, one of the main culprits in the designed QkNN is that for each data point, a new circuit had to be constructed and the state representing the data point had to be initialized on this circuit. This lead to longer runtimes as shown in table 3.1, completely diminishing the theoretical gain that should have been observed by migrating to a quantum computer. A solution to this could be to remove the QRAM method completely and generate real-world data using a General Adversarial Network (GAN). These GANs can be translated to a quantum algorithm (Dallaire-Demers and Killoran 2018, Lloyd and Weedbrook 2018, Zoufal et al. 2019, Huang et al. 2020, Dam et al. 2020) and promise a solution of training a full quantum algorithm using a GAN as a replacement for the resource-expensive QRAM.

Another take on the resource-expensive QRAM is that of Park, Petruccione, et al. (2019a), who have made work of developing a solution to this main drawback. This solution is discussed in further detail by Park, Sinayskiy, et al. (2019b) into what they call "quantum forking", in which they claim to circumvent the redundant state initialization that needs to be done for every new data point for the QkNN. As a result of that, the authors of Blank et al. (2020) have developed a quantum kernel which implements the quantum forking method together with the swap test algorithm to create a classifier. However, as Blank et al. mention, this requires many more qubits which must be able to interact with each other. Still, this is a very promising method from which the QkNN could profit from.

Another method that could be used to fix this is there would exist a way of "burning in" the circuit containing the training data on the quantum circuit. That way, only the register containing the test data and the observational registers (the control qubit and the computational basis $|i\rangle$) would need to be re-initialized, possibly saving lots of compilation time. The first is obvious since there is a new test vector to be classified, but the second is needed since after observation their state has collapsed into the observation, and the superposition needs to be reconstructed. Sadly, at the time of writing, Qiskit does not provide any such method, and the question is if the hardware provided by IBM will ever be capable of such method.

Secondly, a drawback for the QkNN is its variability (or noise). The same solution from Park, Petruccione, et al. (2019a) can be applied here, according to Blank et al. Since they observe the expectation value resulting from the quantum circuit this "opens up a possibility to apply error mitigation techniques to improve the accuracy in the presence of noise without relying on quantum error correcting codes".

Thirdly, as previously stated, the QkNN in this thesis can be considered a hybrid, and further research on implementing the method from Blank et al. (2020) would solve this argument.

Also, as section 1.1 has introduced by quoting Institute of Electrical and Engineers (2017), the data played a large role in the performance of this project. There were three datasets: two out of the box (German credit and Iris) and one manipulated in an attempt to work with all the data (HEP). Since the Iris data was the only dataset that had all its variables used in the classification, it is the one that weighs the heaviest in concluding this project. The addition of the other two datasets is not unimportant. On the contrary, they beautifully show that preprocessing, analysis of the data and knowledge of the data is needed to benefit from the application of machine learning. Since the attention was mostly diverted to the construction and benchmarking of the QkNN, the data science part of this thesis is not as extensive as might have been necessary for the German credit and HEP data. It might be that once properly modified, these two data sets will perform equal to the results shown by the Iris data.

Finally, the kernel has not been executed on a physical quantum computer. The reason for this is that the current NISQ computer is only capable of running circuits with a depth of around 100. As a reference, the minimal circuit depth can be found in figure 3.6, and is roughly 120. It would be interesting to see the different results when executed on a quantum computer. The results would probably vary more than figure 3.4 due to the noise of the quantum computer. The runtimes would probably be lower, however, since the simulated tensor calculus is no longer present because the gate-manipulations are embedded in the hardware itself.

Chapter 5

Conclusion

The main achievement in this thesis is the production and development of a working Quantum k-Nearest Neighbors algorithm, accompanied by a complete documentation and installation guide. This package is now made accessible in an effort to facilitate further development and to encourage reproduction of the results. The results from this thesis show that the QkNN is, however, very limited in its resources. This causes the accuracies to be either low (fig. 3.3) or unstable (fig. 3.4) and the simulated execution times (figs. 3.5 and 3.6) show that the promised reduction in complexity is currently prevented by the limitations of the hardware.

Hence, it is still too early for the QkNN to be commercially available. It is too resource-heavy to be run on larger datasets, does not provide the same accuracy as classically available methods, and has a large skill gap for the user. It does, however, provide a very good reason to keep maintaining the classical methods: proponents of classical computing are stimulated to provide workarounds to outperform theoretical quantum advantages, which is fruitful for everyone.

To advise the use of a QkNN, the accessibility (meaning the SDK, documentation and user interfaces) should first be on a level for a layman to understand and step into. Furthermore, the consistency and scale of quantum computing must be higher, since the noise and limits of qubits are creating a large bottleneck on the results of the QkNN. With these bottlenecks, smart solutions must be invented to have better results one larger data sets, creating a *contradictio in terminis*, since the complexity of the QkNN should scale logarithmically with the dimensionality and size of the data.

Bibliography

Afham, Afrad Basheer, and Sandeep K. Goyal (2020). “Quantum k-nearest neighbor machine learning algorithm”. In: arXiv: 2003.09187 [quant-ph] (cit. on pp. 17–19, 21, 27, 39).

Aleksandrowicz, Gadi et al. (Jan. 2019). *Qiskit: An Open-source Framework for Quantum Computing.* Version 0.7.2. DOI: 10.5281/zenodo.2562111. URL: https://doi.org/10.5281/zenodo.2562111 (cit. on p. 7).

Anderson, Edgar (1936). “The Species Problem in Iris”. In: *Annals of the Missouri Botanical Garden* 23.3, pp. 457–509. ISSN: 00266493. URL: http://www.jstor.org/stable/2394164 (cit. on pp. 22, 24, 32).

Bartley, Christopher (2016). *Replication Data for: German Credit.* Version V1. DOI: 10.7910/DVN/Q8MAW8. URL: https://doi.org/10.7910/DVN/Q8MAW8 (cit. on p. 24).

Benioff, Paul (1980). “The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines”. In: *Journal of statistical physics* 22.5, pp. 563–591 (cit. on p. 6).

Bian, Z. et al. (Nov. 2020). “Fuzzy KNN Method With Adaptive Nearest Neighbors”. In: *IEEE Transactions on Cybernetics*, pp. 1–14. DOI: 10.1109/TCYB.2020.3031610 (cit. on p. 5).

Blank, Carsten et al. (2020). “Quantum classifier with tailored quantum kernel”. In: *npj Quantum Information* 6.8. DOI: 110.1038/s41534-020-0272-6 (cit. on pp. 39, 41, 42).

Boixo, Sergio et al. (2014). “Evidence for quantum annealing with more than one hundred qubits”. In: *Nature Physics* 10. ISSN: 1745-2481. DOI: 10.1038/nphys2900. URL: https://doi.org/10.1038/nphys2900 (cit. on p. 7).

Bracke, Philippe et al. (Aug. 2019). “Machine Learning Explainability in Finance: An Application to Default Risk Analysis”. In: *SSRN*, p. 44. DOI: 10.2139/ssrn.3435104. URL: http://dx.doi.org/10.2139/ssrn.3435104 (cit. on p. 3).

Brassard, Gilles et al. (1998). “Quantum computing”. In: *Proceedings of the National Academy of Sciences* 95.19, pp. 11032–11033. ISSN: 0027-8424. DOI: 10.1073/pnas.95.19.11032. eprint: https://www.pnas.org/content/95/19/11032.

full.pdf. URL: https://www.pnas.org/content/95/19/11032 (cit. on p. 13).

Buhrman, Harry et al. (Sept. 2001). “Quantum Fingerprinting”. In: *Physical Review Letters* 87.16. ISSN: 1079-7114. DOI: 10.1103/physrevlett.87.167902. URL: http://dx.doi.org/10.1103/PhysRevLett.87.167902 (cit. on p. 17).

Burton, Simon (2016). *A Short Guide to Anyons and Modular Functors*. arXiv: 1610.05384 [quant-ph] (cit. on p. 15).

Caron, Sascha, Germán A. Gómez-Vargas, et al. (May 2018). “Analyzing γ rays of the Galactic Center with deep learning”. In: *Journal of Cosmology and Astroparticle Physics* 2018.05, pp. 058–058. DOI: 10.1088/1475-7516/2018/05/058. URL: https://doi.org/10.1088/1475-7516/2018/05/058 (cit. on p. 3).

Caron, Sascha, Tom Heskes, et al. (Nov. 2019). “Constraining the parameters of high-dimensional models with active learning”. In: *The European Physical Journal C* 79.11. ISSN: 1434-6052. DOI: 10.1140/epjc/s10052-019-7437-5. URL: http://dx.doi.org/10.1140/epjc/s10052-019-7437-5 (cit. on p. 2).

Dallaire-Demers, Pierre Luc and Nathan Killoran (2018). “Quantum generative adversarial networks”. In: *Physical Review A* 98.1, pp. 1–8. ISSN: 24699934. DOI: 10.1103/PhysRevA.98.012324. arXiv: 1804.08641 (cit. on p. 41).

Dam, Teresa J. van et al. (June 2020). “Hybrid Helmholtz machines: a gate-based quantum circuit implementation”. In: *Quantum Information Processing* 19.6. ISSN: 15731332. DOI: 10.1007/s11128-020-02660-2 (cit. on p. 41).

DarkMachines (July 2020). *Unsupervised-Hackathon*. DOI: 10.5281/zenodo.3961917. URL: https://doi.org/10.5281/zenodo.3961917 (cit. on pp. 22, 23).

Deutsch, David (1985). “Quantum theory, the Church–Turing principle and the universal quantum computer”. In: *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences* 400.1818, pp. 97–117 (cit. on p. 6).

Diblen, Faruk et al. (2020). “Interactive web-based visualization of multi-dimensional physical and astronomical data”. In: arXiv: 2011.03801 [hep-ex] (cit. on p. 2).

Feynman, Richard P (1982). “Simulating physics with computers”. In: *Int. J. Theor. Phys* 21.6/7 (cit. on p. 6).

Fisher, R. A. (1936). “The use of multiple measurements in taxonomic problems”. In: *Annals of Eugenics* 7.2, pp. 179–188. DOI: 10.1111/j.1469-1809.1936.tb02137.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1469-1809.1936.tb02137.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1469-1809.1936.tb02137.x (cit. on p. 24).

Fredkin, Edward and Tommaso Toffoli (Apr. 1982). “Conservative logic”. In: *International Journal of Theoretical Physics* 21. DOI: 10.1007/BF01857727. URL: https://doi.org/10.1007/BF01857727 (cit. on p. 17).

Freedman, Michael et al. (2003). “Topological quantum computation”. In: *Bulletin of the American Mathematical Society* 40.1, pp. 31–38 (cit. on p. 15).

Gan, Lirong, Huamao Wang, and Zhaojun Yang (2020). “Machine learning solutions to challenges in finance: An application to the pricing of financial products”. In: *Technological Forecasting and Social Change* 153, p. 119928. ISSN: 0040-1625. DOI: `https://doi.org/10.1016/j.techfore.2020.119928`. URL: `http://www.sciencedirect.com/science/article/pii/S0040162519312399` (cit. on p. 3).

García-Gil, Diego et al. (2019). “Enabling Smart Data: Noise filtering in Big Data classification”. In: *Information Sciences* 479, pp. 135–152. ISSN: 0020-0255. DOI: `https://doi.org/10.1016/j.ins.2018.12.002`. URL: `https://www.sciencedirect.com/science/article/pii/S0020025518309460` (cit. on p. 3).

Ghoddusi, Hamed, Germán G. Creamer, and Nima Rafizadeh (2019). “Machine learning in energy economics and finance: A review”. In: *Energy Economics* 81, pp. 709–727. ISSN: 0140-9883. DOI: `https://doi.org/10.1016/j.eneco.2019.05.006`. URL: `http://www.sciencedirect.com/science/article/pii/S0140988319301513` (cit. on p. 3).

Gucht, Jeffrey van der et al. (2020). “Deep Horizon: A machine learning network that recovers accreting black hole parameters”. In: *A&A* 636, A94. DOI: `10.1051/0004-6361/201937014`. URL: `https://doi.org/10.1051/0004-6361/201937014` (cit. on p. 2).

Haber, H.E. and G.L. Kane (1985). “The search for supersymmetry: Probing physics beyond the standard model”. In: *Physics Reports* 117.2, pp. 75–263. ISSN: 0370-1573. DOI: `https://doi.org/10.1016/0370-1573(85)90051-1`. URL: `https://www.sciencedirect.com/science/article/pii/0370157385900511` (cit. on p. 23).

Heaton, J. B., N. G. Polson, and J. H. Witte (2018). *Deep Learning in Finance.* arXiv: `1602.06561 [cs.LG]` (cit. on p. 3).

Hofmann, Hans (1994). *German Credit.* URL: `https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data)` (cit. on p. 22).

Huang, He-Liang et al. (2020). “Experimental Quantum Generative Adversarial Networks for Image Generation”. In: arXiv: `2010.06201`. URL: `http://arxiv.org/abs/2010.06201` (cit. on p. 41).

Institute of Electrical, the and Electronics Engineers (Oct. 2017). “Artificial Intelligence and Machine Learning Applied to Cybersecurity”. In: (cit. on pp. 2, 40, 42).

Kok, Daniël (2020b). “Documentation for Qiskit Quantum kNN: A pure quantum knn classifier for a gated quantum computer.” In: *Read the Docs.* URL: `https://qiskit-quantum-knn.readthedocs.io/` (cit. on pp. 1, 21).

Kok, Daniël (2020a). "Qiskit Quantum kNN: A pure quantum knn classifier for a gated quantum computer." In: *Python Package Index.* URL: https://pypi.org/project/qiskit-quantum-knn/ (cit. on pp. 1, 21).

— (2020c). "qiskit-quantum-knn: A pure quantum knn classifier for a gated quantum computer." In: *GitHub.* URL: https://github.com/GroenteLepel/qiskit-quantum-knn (cit. on pp. 1, 21).

Kumar, M Rajesh, J Venkatesh, and AMJ Md Zubair Rahman (2021). "Data mining and machine learning in retail business: developing efficiencies for better customer retention". In: *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–13. DOI: 10.1007/s12652-020-02711-7. URL: https://doi.org/10.1007/s12652-020-02711-7 (cit. on p. 3).

Liu, Junhua et al. (2019). *Hybrid Quantum-Classical Convolutional Neural Networks.* arXiv: 1911.02998 [quant-ph] (cit. on p. 16).

Liu, Lei and Xinglei Dou (Apr. 2020). "A New Qubits Mapping Mechanism for Multi-programming Quantum Computing". In: (cit. on p. 14).

Lloyd, Seth, Masoud Mohseni, and Patrick Rebentrost (2013). "Quantum algorithms for supervised and unsupervised machine learning". In: arXiv: 1307.0411 [quant-ph] (cit. on pp. 12, 19, 39).

Lloyd, Seth and Christian Weedbrook (2018). "Quantum Generative Adversarial Learning". In: *Physical Review Letters* 121.4, p. 40502. ISSN: 10797114. DOI: 10.1103/PhysRevLett.121.040502. arXiv: 1804.09139. URL: https://doi.org/10.1103/PhysRevLett.121.040502 (cit. on p. 41).

Lykken, Joseph D. (2011). *Beyond the Standard Model.* arXiv: 1005.1676 [hep-ph] (cit. on p. 22).

Mariotto, C. Brenner and M. C. Rodriguez (Sept. 2008). "Investigating gluino production at the LHC". In: *Brazilian Journal of Physics* 38.3b, pp. 503–506. ISSN: 0103-9733. DOI: 10.1590/s0103-97332008000400024. URL: http://dx.doi.org/10.1590/S0103-97332008000400024 (cit. on p. 23).

McClean, Jarrod R et al. (Feb. 2016). "The theory of variational hybrid quantum-classical algorithms". In: *New Journal of Physics* 18.2, p. 023023. DOI: 10.1088/1367-2630/18/2/023023. URL: https://doi.org/10.1088%2F1367-2630%2F18%2F2%2F023023 (cit. on p. 15).

Meister, Smite (2009). *Bloch Sphere.* URL: https://commons.wikimedia.org/w/index.php?curid=5829358 (visited on 10/28/2020) (cit. on p. 10).

Montanaro, Ashley (Nov. 2015). "Quantum algorithms: An overview". In: *npj Quantum Information* 2. DOI: 10.1038/npjqi.2015.23 (cit. on p. 3).

Nakamura, James et al. (2020). "Direct observation of anyonic braiding statistics". In: *Nature Physics* 16.9, pp. 931–936 (cit. on p. 15).

Park, Daniel K., Francesco Petruccione, and June Koo Kevin Rhee (Jan. 2019a). "Circuit-Based Quantum Random Access Memory for Classical Data". In:

Scientific Reports 9.1, pp. 1–9. ISSN: 20452322. DOI: 10.1038/s41598-019-40439-3. arXiv: 1901.02362 (cit. on pp. 41, 42).

Park, Daniel K., Ilya Sinayskiy, et al. (Aug. 2019b). “Parallel quantum trajectories via forking for sampling without redundancy”. In: *New Journal of Physics* 21.8. ISSN: 13672630. DOI: 10.1088/1367-2630/ab35fb. arXiv: 1902.07959 (cit. on p. 41).

Pedregosa, F. et al. (2011). “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12, pp. 2825–2830 (cit. on p. 6).

Rebentrost, Patrick, Masoud Mohseni, and Seth Lloyd (Sept. 2014). “Quantum Support Vector Machine for Big Data Classification”. In: *Phys. Rev. Lett.* 113 (13), p. 130503. DOI: 10.1103/PhysRevLett.113.130503. URL: https://link.aps.org/doi/10.1103/PhysRevLett.113.130503 (cit. on p. 12).

Rosenberg, G. et al. (2016). “Solving the Optimal Trading Trajectory Problem Using a Quantum Annealer”. In: *IEEE Journal of Selected Topics in Signal Processing* 10.6, pp. 1053–1060. DOI: 10.1109/JSTSP.2016.2574703 (cit. on p. 3).

Rosiek, Janusz (June 1990). “Complete set of Feynman rules for the minimal supersymmetric extension of the standard model”. In: *Physical Review D* 41.11, pp. 3464–3501. ISSN: 0556-2821. DOI: 10.1103/physrevd.41.3464. URL: http://dx.doi.org/10.1103/PhysRevD.41.3464 (cit. on p. 24).

Rundo, Francesco et al. (2019). “Machine Learning for Quantitative Finance Applications: A Survey”. In: *Applied Sciences* 9.24. ISSN: 2076-3417. DOI: 10.3390/app9245574. URL: https://www.mdpi.com/2076-3417/9/24/5574 (cit. on p. 3).

Wiebe, Nathan, Ashish Kapoor, and Krysta Svore (2014). *Quantum Algorithms for Nearest-Neighbor Methods for Supervised and Unsupervised Learning.* arXiv: 1401.2142 [quant-ph] (cit. on p. 12).

Wittenbach, Jason, Brian d’Alessandro, and C. Bayan Bruss (2020). “Machine Learning for Temporal Data in Finance: Challenges and Opportunities”. In: arXiv: 2009.05636 [q-fin.ST] (cit. on p. 3).

Woerner, Stefan and Daniel J. Egger (2019). “Quantum Risk Analysis”. In: *Quantum Information.* DOI: 10.1038/s41534-019-0130-6. URL: https://doi.org/10.1038/s41534-019-0130-6 (cit. on p. 3).

Yanofsky, Noson S. (Aug. 2008). *Quantum Computing for Computer Scientists.* Cambridge University Press. ISBN: 0521879965. URL: https://www.xarg.org/ref/a/0521879965/ (cit. on p. 6).

Zhang, Shichao et al. (Jan. 2017). “Learning k for KNN Classification”. In: *ACM Trans. Intell. Syst. Technol.* 8.3. ISSN: 2157-6904. DOI: 10.1145/2990508. URL: https://doi.org/10.1145/2990508 (cit. on p. 4).

Zhao, Puning and Lifeng Lai (2020). *Analysis of KNN Density Estimation.* arXiv: 2010.00438 [stat.ML] (cit. on p. 3).

Zoufal, Christa, Aurélien Lucchi, and Stefan Woerner (2019). "Quantum Generative Adversarial Networks for learning and loading random distributions". In: *npj Quantum Information* 5.1. ISSN: 20566387. DOI: 10.1038/s41534-019-0223-2. arXiv: 1904.00043. URL: http://dx.doi.org/10.1038/s41534-019-0223-2 (cit. on pp. 3, 41).