Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

A Modularised Tool for Quantum/Quantum Enhanced Machine Learning

Ezinwanne Ozoani

September 27, 2021

A dissertation submitted in partial fulfilment
of the requirements for the degree of
MSc (Computer Science)

Declaration

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at `http://www.tcd.ie/calendar`.

I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at `http://tcd-ie.libguides.com/plagiarism/ready-steady-write`.

Signed: _____ Date: _____

Abstract

Machine learning based techniques have become widespread in recent years, and this has driven further developments in the field. One of these developments is the translation of machine learning algorithms from classical computers to quantum computers. The expansion of quantum hardware accessibility enables more extensive research into the potential speed-up quantum algorithms could provide. This speed-up could potentially play a big role in machine learning, where model training becomes slower as the size of training set grows.

The dispersed nature of the resources and the information required to construct a complete quantum machine learning circuit affects the accessibility of quantum machine learning programming. This dissertation provides a centralised resource or starting point for those acquainted with software development rather than theoretical quantum physics. An introduction to the required quantum background, as well as a modular approach to Quantum k-Nearest Neighbour, Quantum Support Vector Mechanism and Grover's Search algorithm is provided. Additionally, these quantum algorithms are tested in order to analyse and observe their quantum speedup claims.

Acknowledgements

Many thanks to my supervisor Dr.Michael Brady from Trinity College for his continuous support and direction on this work. To my loving and supportive family and friends who upheld and encouraged me during my research. Finally, an acknowledgement to the Qiskit and JKU open source community for their input and collaboration.

Contents

List of Figures

1 Introduction

Quantum computers are known to solve certain difficult problems exponentially faster than classical computers. Integer factorisation on a classical computer is one such problem that grows exponentially in time with increasing input data. It can be solved in polynomial time using Shor's algorithm [Shor, 1997] on a quantum computer. Similarly, Grover's Search algorithm, which must be implemented on a quantum computer, provides a quadratic speed-up when searching for an item in an unsorted array when compared to classical search algorithms like heap sort, bubble sort and linear search [Gebhart et al., 2021].

A discipline where quantum computing promises a significant speed-up is Machine Learning (ML). Quantum Machine Learning (QML) or Quantum Enhanced Machine Learning (QEML) has gained a lot of prominence, resulting in an increase in the amount of published literature that covers the principles and techniques of QML [Schuld et al., 2014a]. QML techniques currently in development can solve machine learning problems faster than their known classical algorithms [Ullah Khan, 2019]. QML versions of supervised machine learning techniques such as Support Vector Mechanism (SVM) have been proposed to provide exponential speed-up as compared to their classical counterparts [Abbott and Calude, 2010]. Similarly, QML versions of unsupervised machine learning techniques using clustering and reinforcement have also been proposed.

While quantum computing is not new [Holton, 2021], general access to quantum computing facilities is relatively new. In 2016, the introduction of IBM's Quantum Experience [Moran, 2018] for non institutional users marked the first time general users could readily access quantum or quantum-like[1] cloud systems.

New technologies may be thought of as existing on a "hype cycle" as pictured in Figure 1.1. This curve depicts the current expectations of a technology and its future based on its visibility.

[1] Quantum systems simulated on a classical computer.

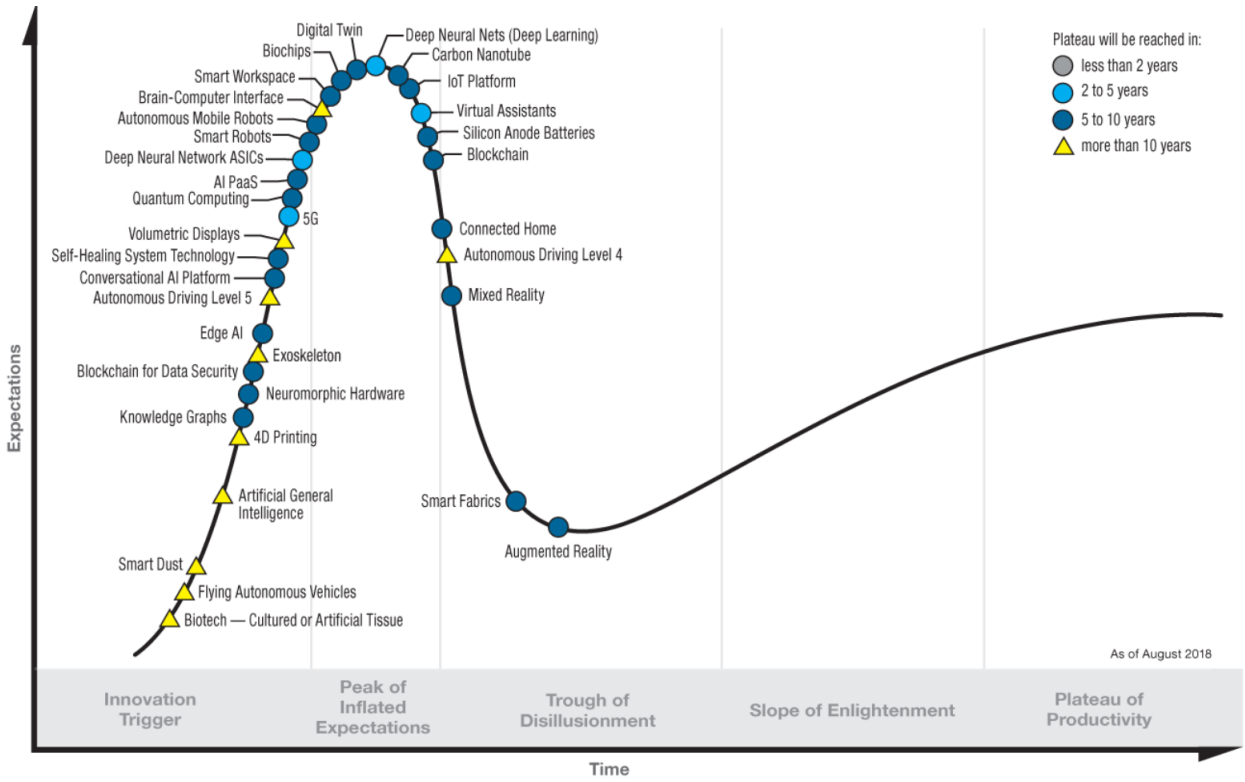

Figure 1.1: The Gartner Hype Cycle for Current Emerging Technologies [Panetta, 2018]

The hype curve has five main stages that can be briefly described as:

1. **Technology Trigger Stage:** The technology trigger stage is when the technology is first being introduced and researched.
2. **Peak of Inflated Expectation:** The peak of inflated expectation is the point at which the technology is being adopted by mass consumers. At this stage, there are high hopes and expectations for future implementations.
3. **Trough of Disillusionment:** At the trough of disillusionment stage, the technology in question is mostly abandoned by mainstream users and they are disillusioned as to its future prospects.
4. **Slope of Enlightenment:** During the slope of enlightenment the 'new' technology is having a slow resurgence and is beginning to show long term potential.
5. **Plateau of Productivity:** The plateau of productivity is the final stage, where the technology does not seem to have any new improvements in prospect, and it appears that future work would be unlikely to contribute significant changes.

It could be said that Quantum Computing is entering into Stage Two of the hype cycle, and that this movement – from a research-centred stage towards more widespread usage – can be attributed to the wider availability of quantum cloud computers and to the increasing

number of publications that include practical implementations. However, there are still few publications that provide a detailed implementation of a quantum machine learning solution[2] from the data encoding stage to the algorithm design stage, through to the analysis of the execution results.

1.1 Quantum Enhanced Machine Learning

A QML or QEML circuit consists of three components: the data encoding, the quantum algorithm and the measurement.

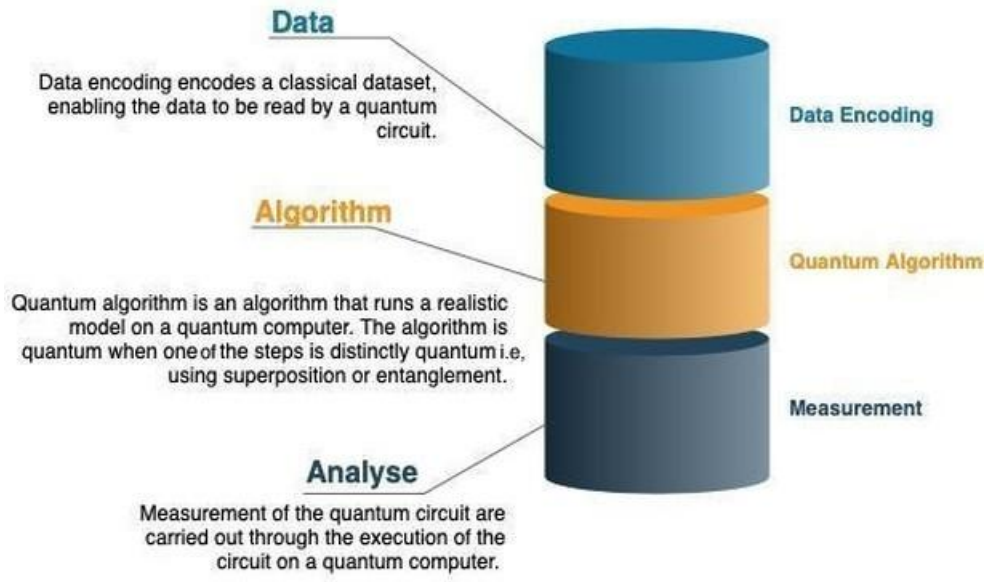

Figure 1.2: General Layout of a Quantum Circuit

In the data encoding component, a classical dataset is encoded into quantum data; this allows the quantum algorithm to read the encoded data as an input. After the execution of the QML circuit on a quantum machine, the output is then measured and analysed.

In this work an analysis of modular implementations of QEML and QML circuits is undertaken, based on the structure illustrated in Figure 1.2. However, before we can discuss the implementations of the different components, we need to familiarise ourselves with the necessary quantum background. In the next section recent publications are reviewed and considered.

1.1.1 Related Works

An influential paper in the field of QML is *An introduction to quantum machine learning* [Schuld et al., 2014b]. This provides a systematic overview of the emerging field of quantum

[2] Commonly referred to as a QEML circuit.

machine learning by presenting approaches and technical details in an accessible manner. While it details Quantum k-Nearest Neighbour (QkNN) and Quantum Support Vector Mechanism (QSVM) algorithms and the concepts behind their design, and while it outlines how to potentially implement them, it does not provide circuit details, circuit implementations using datasets, nor does it analyse QkNN or QSVM circuit outputs. It is rather an introduction that aims to provide an overview of existing ideas and approaches to quantum machine learning.

Another very influential paper is *A Computer Science-Oriented Approach to Introduce Quantum Computing to a New Audience* [Salehi et al., 2021]. This paper is *"an alternative educational approach for introducing quantum computing to a wider audience. It considers quantum computing as a generalised probability theory rather than a field emanating from physics, and it utilises quantum programming as an educational tool to reinforce the learning process."* Similarly to Schuld et al.'s paper referred to above, this paper does not provide code implementations nor does it consider the outputs of quantum algorithms for given datasets. Furthermore, the paper does not explore QkNN, QSVM or any other quantum algorithms. It aims, rather, to *"inform academics and organisations interested in introducing quantum computing to a diverse group of participants from an educational approach"* [Salehi et al., 2021].

Chapter 2 has a similar aim to these research papers, which is to provide an accessible introduction into the background needed for quantum computing for a software engineering audience through a non physics based approach. It is more similar to the approach used in Schuld et al.'s paper, as this chapter examines quantum machine learning algorithms like QSVM and QkNN along with Grover's Search algorithm. Data encoding and output measurements are also explored in this chapter, as detailed Figure 1.2.

Quantum machine learning algorithms can not read classical datasets – they require quantum encoded data. The above publications do not explore quantum data encoding. Papers by Jiang et al. and by Ullah Khan explore quantum data encoding techniques from a theoretical and practical standpoint.

When Machine Learning Meets Quantum Computers: A Case Study [Jiang et al., 2021], demonstrates a framework for implementing neural networks on quantum circuits by providing a data pre-processing implementation using Tensorflow [Abadi et al., 2016], along with neural computation acceleration and data post-processing code for a quantum cloud based computer. Practical code implementations for data encoding are provided, and the MNIST dataset [Deng, 2012] is used to evaluate the data encoding methods. This paper details data encoding through the use of Tensorflow and the `transforms` function in the PyTorch [Paszke et al., 2017] Torchvisions [Marcel and Rodriguez, 2010] library to modify data features. The paper provides a framework for a quantum neural network circuit,

providing code implementations corresponding to the data encoding and the measurement modules in Figure 1.2, where a two layer quantum neural network implementation provides the the quantum algorithm component. Nevertheless, it is just a framework – it does not execute any of the components on a quantum computer, nor does it evaluate their resulting output.

In their thesis entitled *Quantum k Means Algorithm* Ullah Khan *"explores the quantum implementation of K-means clustering algorithm and propose three optimization strategies to achieve shorter-depth circuits for quantum K-means on NISQ [Bharti et al., 2021]* [3] *computers."* [Ullah Khan, 2019]. Amplitude data encoding is used in the quantum implementations of the k Means algorithm to perform clustering using shallow depth quantum circuits. Where Jiang et al. details the use of the MNIST dataset for quantum encoding, this paper employs both the MNIST and the Iris datasets [Lichman, 2013], and evaluates the circuit algorithms on a Qiskit [Koch et al., 2019] quantum cloud based computer. Single cluster and multi cluster quantum k Means implementations are explored in the paper. However, code for these implementations is not provided.

This work will provide the circuit diagrams and the code necessary to implement two variations of data encoding, along with multiple quantum algorithms and modular implementation of these algorithms.

There are publications that provide all three components shown in Figure 1.2, while also making use of circuit diagrams or code snippets, these are *Building a Quantum kNN Classifier with Qiskit: theoretical gains put to practice* by D.J. Kok [Kok, 2021] and *QEML:(Quantum Enhanced Machine Learning) Using Quantum Computation to implement a K-nearest Neighbours Algorithm in a Quantum Feature Space on Superconducting Processors* by Sharma [2020].

Sharma [2020] implements the quantum-enhanced version of the k-Nearest Neighbours algorithm, making use of the same QkNN version implemented in this work, the Hamming distance. This paper also explores multiple datasets namely, the Iris and the Wisconsin Cancer datasets. However, it only evaluates the QkNN circuit using 10 data points from these datasets, as the method for data encoding, maps each datapoint to a quantum gate. This encoding method restricts the number of data points that can be used, based on the current number of qubits available on a quantum computer. [4] Sharma also evaluates the QkNN circuit on a quantum simulator, instead of using a real quantum device.

Kok implements a version of QkNN capable of performing kNN classification using the Dot

[3] Noisy intermediate-scale quantum (NISQ) algorithms. We are far from a universal fault-tolerant quantum computer, it will potentially take decades to reach that standard. NISQ computers are here right now, though they have hundreds of noisy bits, they can leverage limited quantum resources in order to preform classically challenging tasks.

[4] The current largest quantum machine that is open to non institutional use is limited to 15 qubits.

Product method. Similar to [Sharma, 2020], this paper makes use of the Iris dataset, but also the Hofmann (German Credit dataset) and the HEP datasets. It also incorporates more data points into the output evaluation as it makes use of the analog encoding technique, which normalises the data points and allows for data to be encoding into a smaller number of qubits. However, both Sharma's paper and Kok paper evaluates the QkNN circuit using only a simulated quantum device, this paper executes the quantum circuit on a quantum cloud based computer.

While both publications detail QML implementations for a singular quantum algorithm and data encoding technique, Kok only implements QkNN using the Qsiksit built-in function. This work incorporates the Qiskit built in function but it also provides the Hamming distance circuit implementation found in Sharma. These QkNN implementations are illustrated in Section 4.2.1 as part of the modular design for the circuit, along with two implementations for data encoding that are similarly evaluated using multiple datasets.

1.2 Motivation

When one first begins to study quantum computing with the aim of writing quantum programs, one will learn about: the different quantum machine learning and quantum based algorithms, how to run a circuit on a quantum machine and possibly data encoding techniques from one of the aforementioned publications. The ability to see all the necessary stages and different components that could be implemented is not centralised.

Taking the first stage – data encoding – most publications and bodies of work, including those mentioned, do not provide both the code needed for implementing them and their circuit illustrations. Finding both of these for more than one data encoding technique and for more than a single quantum machine algorithm, was not found during the research stage of this dissertation. Taking the quantum library Qiskit as an example, particularly their documentation and instructional videos, it can be noted that the documentation provides function calls for data encoding, while the videos focus on different quantum algorithm implementations with code that calls their built in "black box" functions. It does not detail, within the same resource location, how to incorporate all the stages shown in Figure 1.2 into a full QML circuit, or how each of these components operate.

The intention of this dissertation is to deliver the various stages in a quantum system by implementing the various theoretical research methods that can be found across different publications, and by presenting them in a layout that can be configured with these components in a modular manner. Thus providing software engineers, who may be new quantum users, with the ability to combine different data encoding techniques with different quantum algorithms as they learn. It also provides a modular foundation for more quantum

algorithms, such as k Means, to be added to the modular circuit and tested with the existing modular data encoding components or, for additional data encoding techniques such as digital encoding to be incorporated in the same manner.

1.3 Method

The publications explored provide partial or complete stages of a QML circuit, only focused on the implementation of one component of each. As illustrated in Figure 1.3, this work will explore each step of a quantum circuit and provide options for each stage and the modular code needed for their implementation.

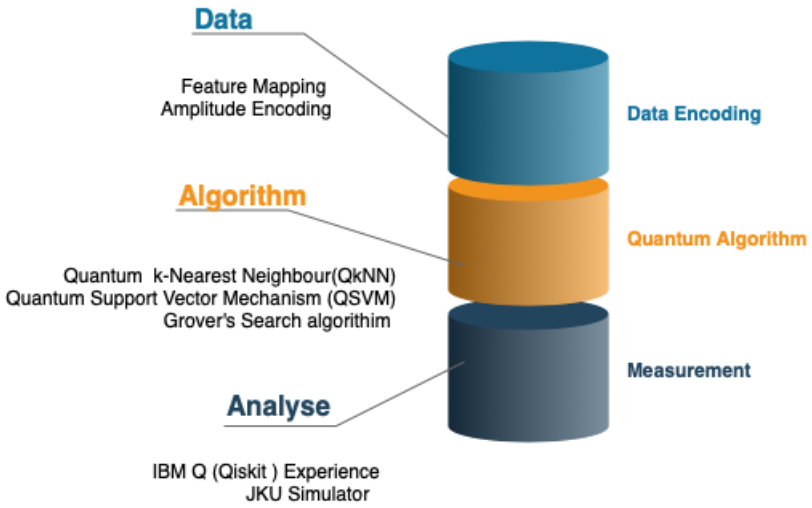

Figure 1.3: Layout of a Quantum Circuit Components Implemented

To achieve this, the following contributions are made in this dissertation:

- The first and the most important aim is to provide a circuit with all three stages found in Figure 1.3 with modular components in an accessible form. This enables future development on this work to be easy and approachable, particularly for quantum enthusiasts and those more literate in software development than in quantum physics. To do so, the necessary python code is presented in a technical manner where detailed knowledge of quantum physics would not be necessary. However, the basic quantum theoretical knowledge will be explained in a concise and austere style.
- For the algorithm stage of the circuit, a full quantum machine learning implementation of k-Nearest Neighbour is detailed based on the descriptions detailed in the work *Algorithm for k-Nearest Neighbours Classification based on the Metric of Hamming Distance* (12) [Sharma, 2020]. Two variations of Grover's Search algorithm and QSVM are also explored, these are delivered through illustrative and succinct implementations.

- As previously stated, in order to run classical data on a quantum circuit, quantum readable data is required. Existing data encoding methods, both theoretical and implemented, are dispersed across various research publications. This work presents different methods for data encoding both the circuit design and the code in a centralised location.
- Finally, bench-marking is performed on the aforementioned quantum machine learning algorithms and their classical counterparts. Similar to the publications we explored, the circuit is analysed using multiple datasets with various data points, executed on a quantum computer. In addition to the quantum computer execution, this work details an external classical quantum based simulator, the JKU simulator, its advantages, current state and the necessary implementation. These measurement techniques allow for a critical review of the circuit results, for the different circuit components.

2 Background

This chapter will review the necessary concepts of quantum computing that are required in order to further explore the modular approach. A circuit consists of data, operations and results, these components provide the context for the following segment.

2.1 Qubit

Similar to bits found in classical computers, a quantum bit or qubit is the smallest unit of measurement in a quantum computer. Using the Dirac Vector notation or the Bra-Ket notation [Yisroel Meir Blumstein, Kalle Vedin, Tom Reding, Sammi Brie, Dr James V Stone, Peter Ells., 2004], we can represent a qubit in state zero as $|0\rangle$ with $|1\rangle$ for a qubit in state one. These can be denoted in vector form as:

$$|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \qquad |0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{1}$$

Two qubits can be represented in a four-dimensional linear vector space that holds the probability amplitudes of each possible state. These are spanned by the following product basis states:

$$|00\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, |01\rangle = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, |10\rangle = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \text{and } |11\rangle = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}.$$

Figure 2.1: Two Qubit Vector Representation [Yisroel Meir Blumstein, Kalle Vedin, Tom Reding, Sammi Brie, Dr James V Stone, Peter Ells., 2004]

Bloch Sphere

The geometrical representation of a qubit is usually illustrated using the Bloch sphere. The Bloch sphere is a two level system consisting of a northern and southern hemisphere. The north pole of the sphere is assigned the state $|0\rangle$ and the south pole the state $|1\rangle$. A classical bit would be represented on the Bloch sphere as being in either the north pole of

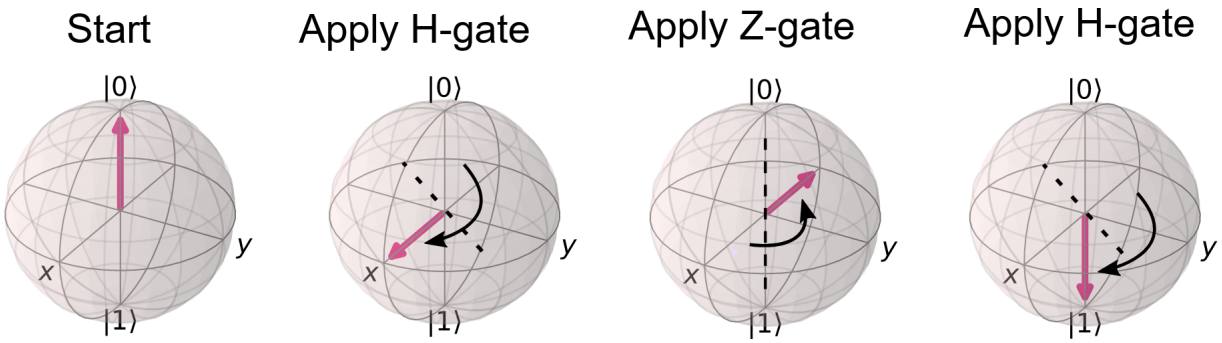

Figure 2.2: Transcription within Bloch Sphere with respect to gate operation Community [2021]

the sphere or the south pole. A qubit however, can be a point anywhere on the surface of the sphere [hÉigeartaigh, 2003].

When encoding classical data into a quantum space, a data point is represented as an angle. Referring to Figure 2.2, the angle is then mapped to a coordinate within the Bloch sphere. However, the Bloch sphere is not a precise indicator of where a qubit lies on the unit sphere, it merely shows the latitude of the qubit. The latitude defines how close the qubit is to the poles, depending on the probability amplitudes [hÉigeartaigh, 2003].

2.1.1 Entanglement and Superposition

Superposition

Classical bits always have a completely well-defined state: they are either 0 or 1 at every point of a computation. The difference between a qubit and a classical bit is the possibility for a qubit to be in a linear combination of states, denoted by;

$$|\phi\rangle = \alpha|0\rangle + \beta|1\rangle$$

The values α and β are complex numbers and their ability to exist in multiple states is called superposition.

Entanglement

Quantum machines do not allow the amplitudes of α and β to be directly viewed. A qubit can therefore encode an infinite amount of information, but most of this information is useless as it can never be observed [hÉigeartaigh, 2003]. In order to observe the state that a qubit is in, a measurement is performed.

Taking a two qubit system, there are four basis states that they can collapse to, which are $|00\rangle$, $|01\rangle$, $|10\rangle$ and $|11\rangle$. These states are referred to as the Bell States or EPR states

[hÉigeartaigh, 2003]. An example of one of these states is the following (from hÉigeartaigh [2003]):

$$|\phi\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{(2)}}$$

When the first qubit is measured, there are two possible results:

If the first qubit is in state 0, it provides the probability of the other qubit residing in state $|00\rangle$ as 1/2.

If the first qubit is in state 1, it also produces the probability of the other qubit residing in state $|11\rangle$ as 1/2.

This means that when the second qubit is measured it will always be in the same state as the first qubit. This correlation between the qubits is known as entanglement [hÉigeartaigh, 2003].

2.1.2 Quantum Gates

Quantum logic gates or simply quantum gates are the building blocks of a quantum circuit. They can be used to manipulate and transform qubit states. Quantum gates are briefly described here, for a more in depth treatment of this topic please refer to *Quantum computing for computer scientists* [Yanofsky and Mannucci, 2008].

Hadamard gate or the H gate: The Hadamard gate is defined by the unitary matrix:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad \text{—}\boxed{H}\text{—}$$

Figure 2.3: Hadamard Gate [Ullah Khan, 2019]

It is objectively one of the most important gates, as it serves to place qubits in equal superposition of $|0\rangle$and $|1\rangle$.

NOT gate or the X gate: The NOT gate is the quantum equivalent of the NOT gate found in classical computers. It flips the probabilities of state $|0\rangle$ and $|1\rangle$.

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Figure 2.4: NOT Gate [Ullah Khan, 2019]

Identity gate or the ID gate: The Identity gate is a single-qubit operation that leaves the basis states $|0\rangle$ and $|1\rangle$ unchanged. It ensures that no operation occurs on the qubit for one unit of time.

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Controlled NOT or CNOT gate: The CNOT gate acts on two qubits and performs the NOT operation on the second qubit only when the first qubit is $|1\rangle$, otherwise it leaves it unchanged [Yisroel Meir Blumstein, Kalle Vedin, Tom Reding, Sammi Brie, Dr James V Stone, Peter Ells., 2004].

$$CX = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

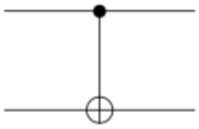

Figure 2.5: Controlled NOT Gate [Ullah Khan, 2019]

Toffoli (Controlled Controlled Not gate [CCNOT]): The Toffoli gate is a three-qubit gate in which two qubits act as a control and the third qubit is the target. If both of the control qubits are in state $|1\rangle$ then the target qubit is flipped and the control qubits remain unchanged [Ullah Khan, 2019].

$$CCX = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

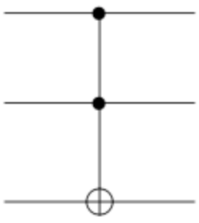

Figure 2.6: Toffoli Gate [Ullah Khan, 2019]

Rotation Y or Ry gate: The Ry gate rotates the qubit along the y-axis to the specified angle Θ within the Bloch sphere. This can be useful for mapping classical data points into a quantum space.

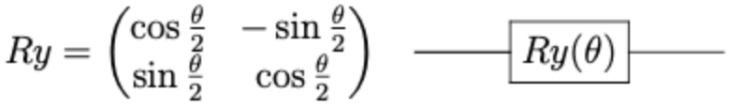

Figure 2.7: Rotation Y Gate [Ullah Khan, 2019]

SWAP gate: The SWAP gate is a two-qubit operation that swaps the state of the qubits involved in the operation.

Figure 2.8: SWAP Gate

As shown in Figure 2.9, the SWAP gate allows us to move information around in a quantum circuit without effecting the qubit data.

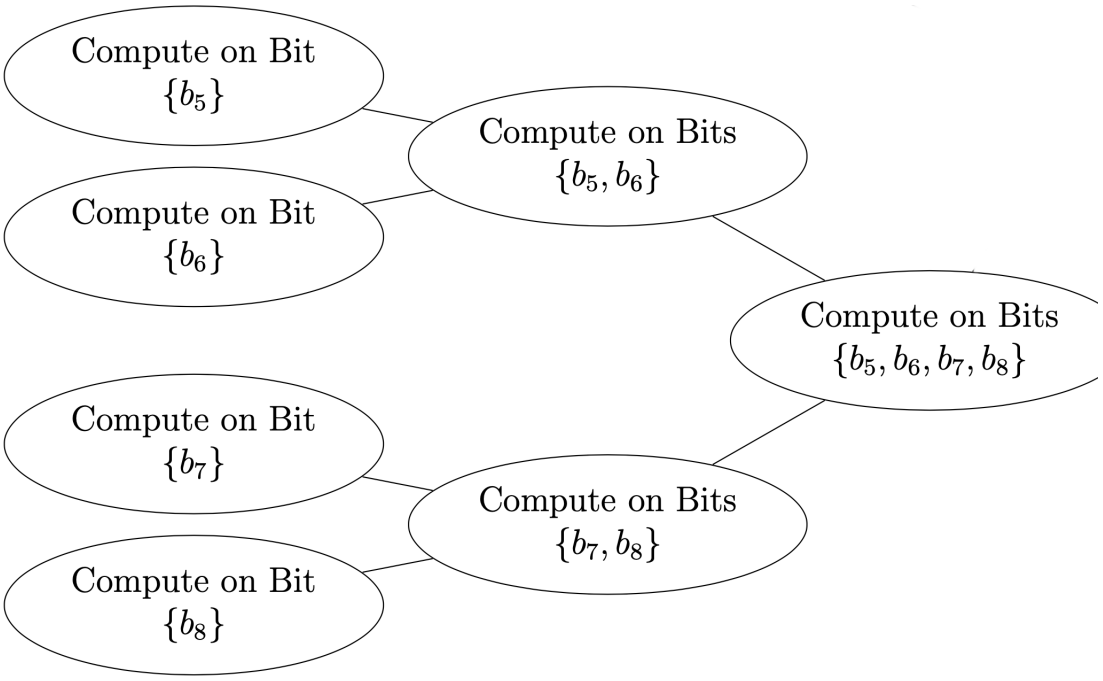

Figure 2.9: SWAP Gate Process [Cortese and Braje, 2018]

2.2 Quantum Data Encoding

With the knowledge of how bits are represented, an understanding of how classical data is stored on quantum bits is needed as encoding data in qubits is not trivial. Current quantum devices contain a limited amount of qubits that are stable for a short amount of time

[Weigold et al., 2021]. In order to make use of current quantum devices our circuit representation must be compact and limited to a set amount of qubits and quantum gates. To do so we will exercise quantum data encoding or data embedding. These techniques allow us to take a classical data point, x, and encode it by applying a set of gate parameters on the quantum circuit. The operations that will be performed depend on the value of x, hence creating the desired quantum state.

There exist different methods for data encoding, an example of these are:

Basis Encoding: This is categorised as a digital encoding technique because it is suitable for arithmetic computations [Leymann and Barzen, 2020]. In this method, data is simply encoded into binary strings, where each input is converted to a computational basis state of a qubit system. While this provides a lot of freedom computationally, the available implementation schemes are usually not efficient [David, 2021].

Angle Encoding: In this type of encoding, classical information is encoded into angle rotations of a qubit. This results in using the feature values of an input data point, x, as angles in a unitary quantum gate [David, 2021].

This discourse makes use of amplitude encoding and feature mapping to encode classical data.

2.2.1 Amplitude Encoding

Amplitude Encoding can also be referred to as Wavefunction Encoding [LaRose and Coyle, 2020]. It consists of mapping the coordinates of a vector into the values of the amplitudes of a quantum state. It requires the vector to be normalised and to have a power-of-two dimensions [Thabet, 2020].

Section 4.1.1 implements an example of when these conditions are not fulfilled by the dataset, resulting in the need to pad and re-normalise the vector. This is carried out by preparing the data through normalisation and feature padding; the datapoint angles are then extracted from this prepared data. The extracted angles are then rotated to encode them within the Bloch sphere and this encoding allows the data to be encoded into a quantum state.

This type of data encoding is very useful for smaller datasets with multiple features, as it does not map each feature to a specific qubit. However, it does take multiple steps to encode the data, and with larger datasets this could increase resource requirements.

2.2.2 Feature Mapping

A feature map reduces the amount of resources required to describe a large set of data as it maps this data into a higher dimensional space or quantum Hilbert space [David, 2021].

Hilbert Space: The Hilbert space is a large space where the states that describe quantum systems live. For a 50-qubit quantum computer, it would be a 1,125,899,907,000,000 - dimensional space. For a single mode of a continuous-variable quantum computer, the Hilbert space has an infinite number of dimensions [Schuld, 2018].

Feature maps encode the classical data by applying the parameterised circuit $V(\phi(\overrightarrow{x_i})$, which converts the classical data into quantum data. Where:

$\overrightarrow{x_i}$: Represents the classical data set.

$\phi()$: Is the classical function applied on the classical data set.

V: Is the vector space.

As discussed earlier in Section 2.1, qubits can be denoted in vector form. Feature maps use this vector notation to encode the classical data x_i into quantum states.

Section 4.1.2 of this dissertation implements three different types of pre-coded feature maps found in the Qiskit circuit library, namely: ZZFeaturemap, ZFeaturemap and PauliFeaturemap. It is possible to vary the depths of these feature maps (1, 2, 4) in order to check a models performance. By increasing a feature map's depth, we introduce more entanglement into the model [David, 2021].

2.3 Quantum Circuit Application

This section builds on the knowledge of circuit components and data encoding by utilising them to express operations. It first needs to be established, which quantum operations or circuits one wishes to implement.

2.3.1 Quantum k-Nearest Neighbour (QkNN)

The k-Nearest Neighbour algorithm is a simple supervised machine learning algorithm that is used extensively for pattern recognition and classification [Harrison, 2018]. When given a testing sample, the algorithm find its k nearest neighbours based on some distance metric and then determines its category according to the information of these neighbours.

There exists different quantum k-Nearest Neighbour implementations, one such example, that is explored later in this work, is the Hamming Distance.

Quantum k-Nearest Neighbour Algorithm Using Fidelity

QkNN using *fidelity* [Basheer et al., 2020] performs a SWAP test between the test states and all training states in superposition to analog-encode the fidelity. The SWAP test is a quantum algorithm that can be used to statistically estimate the fidelity of the two pure states $|\psi\rangle$ and $|\phi\rangle$ as follows:

1. An initial preparation of three registers in states $|0\rangle$, $|\psi\rangle$, $|\phi\rangle$ is needed in order to implement the SWAP test. This results in an initial combined state of the three registers as: $|R\rangle = |0\rangle \otimes |\psi\rangle \otimes |\phi\rangle$ [Jafarizadeh et al., 2020].
2. Then a initial Hadamard operation is applied on the first register followed by a control SWAP on the other two registers, with the first register serving as the control in the system Jafarizadeh et al. [2020].
3. Lastly, another Hadamard operation (H) is applied on the first qubit $|0\rangle$, $|1\rangle$, resulting in 0 and 1 with the probabilities:

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

Figure 2.10: Quantum K-Nearest Neighbour Using Fidelity [Jafarizadeh et al., 2020]

The quantity $P(0) - P(1)$ gives us the desired fidelity [Jafarizadeh et al., 2020].

Quantum k-Nearest Neighbour Algorithm Using Hamming Distance

Section 4.2.1 implements the Hamming distance QkNN as described in *Quantum Algorithm for K-Nearest Neighbours Classification Based on the Metric of Hamming Distance* [Ruan et al., 2017].

After the necessary data encoding steps, the condition of the Hamming distance is sought. The Hamming distance is defined as counting the differing number of corresponding symbols in two bit-vectors of equal length e.g:

Hamming Distance: 0110 $\leftrightarrow$ 0001, has a distance of 3.

Hamming Distance: 0110 $\leftrightarrow$ 1110, has a distance of 1.

The condition of the Hamming distance is retrieved by applying two quantum additions and a quantum OR operation to the most significant bit. This results in a zero or one output, which indicates the presence of a neighbour.

When using the classical kNN algorithm on larger data sets, there is a noticeable degradation in the performance of the algorithm. Quantum k-Nearest Neighbour places the

entire the training set into superposition at the very start. Thus, in one run it can calculate all distances between the input and the entire training set. This allows the circuits to only require as much qubits as is needed to encode the entire training set.

2.3.2 Quantum Support Vector Mechanism (QSVM)

Support Vector Mechanism(SVM) is a form of supervised learning used to classify data into categories. SVM takes a training dataset and virtually plots the data in d-dimensional space, where d is the number of parameters or features of each data point. It then attempts to find the (d-1)-dimensional hyper-plane which separates the data into two classes [R., 2003].

In many cases it is impossible to draw a separating hyperplane between classes of data, especially when the boundary is highly nonlinear [R., 2003]. In such a case where the SVM is unable to find a suitable separating hyperplane, a technique known as a kernel trick is used [McRae and Hilke, 2020b]. The kernel rick uses a nonlinear function or a feature map to project the data into a higher dimension where a separating hyperplane may be found [Boser et al., 1992].

In the case of the Quantum Support Vector Mechanism(QSVM) used by *[McRae and Hilke, 2020a]*, only the quantum feature maps $V(\phi(\overrightarrow{x_i}))$ is used to translate the classical data $\overrightarrow{x}$ into quantum states. The kernel for the SVM is built out of these quantum states. After calculating the kernel matrix using quantum mapped data, the quantum SVM can then be trained in the same fashion as the classical SVM [Huembeli, 2019].

The idea of the quantum kernel is exactly the same as the classical case where the inner product of the feature map $K(\overrightarrow{x}, \overrightarrow{z}) = |\langle\Phi(\overrightarrow{x})|\Phi(\overrightarrow{z})\rangle|^2$, is taken but with the quantum feature mapped data in replacement.

Section 4.2.2 will reveal how to make use of the predefined functions provided by Qiskit Aqua to implement QSVM. These functions will require the feature map, a training and a test set as inputs, in order to train the whole QSVM.

QSVM Quantum Advantage

The quantum version of the SVM is best described as "quantum-assisted" or "quantum-enhanced" [Arunachalam and de Wolf, 2017], in the sense that the algorithm is largely classical with certain operations performed by a quantum processor.

In the case of a quantum SVM, the quantum feature maps are used to translate the classical data into quantum states. The classical kernel used for SVM is then built out of these quantum states, which are then used to train the QSVM in the same way as the classical SVM.

QSVM will minimise the loss via optimising the parameters. However, apart from finding the quantum kernel data, the QSVM algorithm preforms only classical optimisation. In the end there is no significant difference between the QSVM and the classical SVM.

2.3.3 Grover's Search Algorithm

This section will introduce Grover's Search algorithm and how it can be used to solve unstructured search problems.

Grover's Search algorithm is named after the creator Lov Kumar Grover. It is a quantum computing algorithm that can search databases much faster than a classical computer. Grover's Search algorithm can speed up an unstructured search problem quadratically. It is one of the first and most prominent examples to showcase how a quantum circuit can be magnitudes faster than a classical algorithm [Zickert, 2021]. As it takes $O(N1/2)$ time, while a linear search, which is $O(N)$ time. As such, it is the fastest possible quantum algorithm for searching an unsorted database.

Although the purpose of Grover's Search algorithm is usually described as searching a database, it may be more accurate to describe it as inverting a function. Taking a function $y = f(x)$, that can be evaluated on a quantum computer, Grover's Search algorithm allows for the calculation of x when given y. Inverting a function is related to the searching of a database as it can create a function that produces a particular value of y if x matches a desired entry in a database, and another value of y for other values of x [Team, 2015]. Borbely *Grover Search algorithm* [Borbely, 2007], provides a more in-dept exploration into the inner workings of Grover's Search algorithm.

Grover's Search algorithm can also be used for estimating the mean and median of a set of numbers, and for solving the collision problem. In addition, it can be used to solve NP-complete problems by performing exhaustive searches over the set of possible solutions.

The application of Grover's Search results in "only" a quadratic speedup, unlike other quantum algorithms, which can provide exponential speedup over their classical counterparts. However, this quadratic speed-up becomes considerable when N is large.

2.4 Quantum Computers

The final step in quantum circuit as illustrated in Figure 1.3 is the measurement step. If we think of Schrödinger's cat, which can be dead or alive with some probability, opening the box is "measuring" the state of the cat. In the same way, quantum computers estimate the probability of a qubit belonging to a class by performing certain measurements.

To carry out this measurement would require executing our circuit on a quantum computer. However, quantum computers are expensive and costly to maintain, as such we will make use of cloud-based quantum computers.

Cloud-based quantum computing allows companies, researchers and individuals to test their quantum algorithms. Cloud-based quantum computing achieves this by providing direct access to emulators, simulators and quantum processors. Vendors also provide development platforms and documentation for quantum computing languages and tools [Dilmegani, 2021a].

2.4.1 Quantum Machines

Cloud-based quantum machines provide an environment for businesses and academia to practice quantum approaches without having to wait for quantum computing technology to mature and become more widespread [Dilmegani, 2021a].

Here are some of the more prominent cloud-based quantum computing providers:

D-Wave Systems: Founded in 1999, D-Wave is one of the earliest players in quantum computing. They were the first company to provide a commercially available quantum computer. However, the D-Wave machine has very weak coherence times, and the range of Hamiltonians[1] they can produce is limited. This results in the D-Wave device(s) being more suited to use as a quantum enhanced annealer [Dilmegani, 2021a].

Quantum annealing is a quantum computing method used to find the optimal solution of problems involving a large number of solutions, by taking advantage of properties specific to quantum physics like quantum tunneling, entanglement and superposition [Dilmegani, 2021b]. It is often used, in preference to other methods, to find the global minimum of a given object function.

In the current research landscape, quantum annealing is used for optimisation machine learning problems in four broad areas [Nath et al., 2021]:

(i) Image Recognition

(ii) Remote Sensing Imagery

(iii) Particle Physics

(iv) Computational Biology.

It is noted that *"Machine learning tasks using quantum annealing are performed in a hybrid classical and quantum system"*. The use of D-Wave's quantum annealing system would

[1]The Hamiltonian of a system is an operator corresponding to the total energy of that system, including both kinetic energy and potential energy. [El C, ReyHahn, Le Deluge, , 2019]

allow for the implementation of a hybrid QSVM algorithm but the implementation of a full quantum QkNN algorithm may not be currently feasible [Nath et al., 2021].

Google's Quantum Playground: Google's Quantum Playground provides a simulator with a user interface, scripting language and 3D quantum state visualisations. Along with this, Google announced the achievement of quantum supremacy by using a 54-qubit Sycamore processor in late 2019 [Dilmegani, 2021a]. Although the platform offers access to TensorFlow Quantum, which is a library for building quantum machine-learning models, they have not yet included a general-purpose quantum computing service.

Microsoft Quantum Computing: Microsoft provides Quantum Development Kits (QDK) and a quantum script language called *Q#* for quantum computing development. *Q#* is a C−style programming language consisting of a *"set of libraries that abstract complex functionality in Q#, APIs for Python and .NET languages (C#, F#, and VB.NET) for running quantum programs written in Q#, and tools to facilitate your development"* [LaRose, 2019]. Microsoft partnered with 1Qbit, Honeywell, IONQ and QCI during the development of their quantum computing systems. Microsoft has also developed their own quantum system called Station Q [Dilmegani, 2021a].

The Microsoft QDK was not used for this dissertation implementation because it lacks support for a SWAP gate to move quantum information around a quantum circuit [LaRose, 2019]. Additionally it could be argued that Python is a more approachable language than a C−style language such as *Q#* .

IBM Q Experience: IBM introduced a quantum network called IBM Q network in 2016. Since then, IBM became one of the forerunners in the quantum computing ecosystem. IBM Q can be accessed on the cloud through Qiskit, which is an open-source quantum software development kit [Dilmegani, 2021a].

This work will make use of IBM's Qiskit network. While Qiskit does not support quantum annealing like the D-Wave system and it has not reached quantum supremacy like Google, Qiskit has many other advantages. These include having a large and active open source community, its documentation is slightly more centralised and Qiskit supports third party simulators, which are all necessary components for this modular tool.

2.4.2 Errors, Noise and Interference

Unlike classical computers which are known for their predictable determinism, quantum computers are inherently probabilistic by nature [Krupansky, 2020]. This section will explore how quantum computers are made even less predictable through quantum errors, noise, interference and decoherence.

Quantum Errors

Errors can be caused by noise and environmental interference. However, they are generally caused by the flawed execution of operations, such as quantum logic gates and measurement operations, as well as decoherence [Krupansky, 2020]. There also exists spectator errors or crosstalk, this is when an operation on one qubit can have an unintended effect on a nearby qubit. Using ID gates on qubits that one does not intended to have operations running on, can slightly reduce this possibility. However, this will only aim to reduce these types of errors and not mitigate them from existing within the quantum process.

Quantum Noise

When implementing a quantum algorithm it is important to consider the sources of the noise.

Noise differs from environmental noise in that the source of the noise comes from within the quantum machine itself. These include electrical and mechanical components, cabling, wiring, circuit connections, and intermittent interactions of any of the preceding [Krupansky, 2020]. Environmental noise can stem from *"uncontrolled interactions between system and environment, such as thermal motion of charge carriers or stray magnetic fields, lead to deviations between the targeted and the actual evolution"* [Suter and Alvarez, 2016]; which in turn can lead to a loss of coherence in the system.

The two main sources of noise are typically gate infidelity and decoherence.

Gate Infidelity: The more qubits present in a system, the higher the probability that one qubit will loose too much information, this can cause the entire circuit to become useless. The number of qubits required to execute a quantum circuit depends on the complexity of the circuit. Therefore, large quantum circuits constructed using many qubits are more susceptible to noise [Kok, 2021].

Quantum Interference and Decoherence

Quantum machines can only predict the probability of a certain outcome, with the final probability being assumed during the measurement stage. Quantum interference is a byproduct of superposition, in that it can be used to reinforce the probability of obtaining the desired result, all while reducing unwanted results via constructive or destructive interference [Kok, 2021]. A disruption in quantum interference can be a source of error and this type of disruption is called decoherence.

Decoherence: Decoherence refers to the fact that gradually over time a quantum computer has the probability to behave more like a classical object. After decoherence has fully occurred, the computer can no longer take advantage of quantum effects and this introduces

progressively more noise as the quantum algorithm proceeds [Coles et al., 2018].

Error, Noise and Interference Correction

There are methods available that can reduce error, noise and interference.

Shots: When executing our quantum circuit we can specify the number of shots. These shots are effectively the number of repetitions the circuit will undertake with the intention that environmental interference and possibly noise and errors will be removed to some degree.

Quantum Error Correction: Quantum Error Correction (QEC) is used to protect quantum information from errors that are primarily due to decoherence and quantum noise. Quantum error correction is essential in order to achieve fault-tolerant quantum computation that can deal not only with noise found in stored quantum information but also with faulty quantum gates, faulty quantum preparation and faulty measurements [Wikipedia, 2021].

While QEC is in the early stages of development, it acts by protecting the information stored in qubits by not storing this information in individual qubits but rather in patterns of entanglement among many qubits. More information can be gathered about quantum error correction in *Simon J. Devitt, William J. Munro, and Kae Nemoto work, Quantum Error Correction for Beginners* [Devitt et al., 2013].

2.4.3 Quantum Simulators

Quantum simulators enable us to simulate both quantum inherent algorithms such as Grover's Search and the quantum counterpart of classical machine learning algorithms e.g QkNN, without loosing the benefits that a quantum environment supplies and in an ideal quantum space without noise.

The JKU Simulator: Qiskit is designed to include third-party simulators that are usually contributed by the quantum open source community. A prime example of such a simulator is the JKU simulator created by Alwin Zulehner and Robert Wille from the quantum computation team at Johannes Kepler University in Linz, Austria [Naveh, 2019].

This simulator stores quantum states using a data structure which is based on classical decision diagrams. This representation is more complex than simply storing a state vector, rather the vector has regular multiplicities, which then results in a much more efficient storage space and manipulation time [Naveh, 2019].

3 State of The Art

This chapter analyses previous work related to quantum computing and quantum enhanced machine learning. We examine publications that are research based on the one hand and we also examine publications focused on implemented approaches. We compare those with implementations developed in this dissertation.

Section 3.1 investigates the relevant work that provides the foundation for quantum machine learning. These publications lay the groundwork for quantum machine learning algorithms through a physics based approach. Moving on, Section 3.2 examines publications that propose implementations but provide different levels of detail.

It is hoped that the structures and implementations developed in this dissertation will facilitate a more thorough understanding of these techniques by readers from a computational background. Through the use of current data encoding methods and the greater accessibility of quantum machines, these groundwork techniques can be applied and their full potential and quantum advantages assessed.

3.1 Groundwork

[Schuld et al., 2014b]'s *An introduction to quantum machine learning* provides an overview of the field of quantum machine learning by outlining quantum machine learning algorithms. It provides an analysis of the different quantum approaches that are currently being explored and the potential these approaches have in relation to quantum machine learning. A range of quantum methods such as: quantum machine learning algorithms, quantum neural networks, quantum classification using Bayesian decision theory and hidden quantum Markov models are discussed. Each topic is presented by introducing their classical version and their significance in machine learning. Following this, their quantum counterparts are introduced and an overview of current work is provided. A selection of relevant papers are explored particularly with respect to the implementations developed therein.

The quantum machine algorithms QkNN and QSVM developed in the paper are of particular interest. Two different QkNN approaches, discussed in [Schuld et al., 2014b] are detailed in

the following works: *Machine Learning in a Quantum World* [Aïmeur et al., 2006], *Quantum algorithms for supervised and unsupervised machine learning* [Lloyd et al., 2013] and *Quantum Pattern Recognition* [Trugenberger, 2002].

The idea of using the *SWAP test* is introduced in Aïmeur et al. and Lloyd et al., the *SWAP test* is an inexpensive procedure that estimates the fidelity or overlap in a quantum routine. The *SWAP test* uses the C-*SWAP* test several times on the training set, which encodes the classical length of the vector data into a scalar product for the quantum state, in order to estimate the fidelity between each pair in the set. This allows for the retrieval of the classical distance between two points. As explained in Section 2.3.1, the *SWAP test* is not a true quantum approach, it is instead a quantum enhanced QkNN approach, this is due to its use during the initial clustering, after which a classical clustering algorithm such as k-means is applied on the data that is obtained. This approach allows for the addition of a new data point, as the k-Means latter half finds the nearest centroid for this new input. However, Schuld et al. noted both of these papers claim that the *SWAP test* QkNN approach is more efficient than the polynomial run-time of a classical QkNN approach, but they do not make use of a quantum system or a quantum circuit in order to evaluate their results.

The third paper, [Trugenberger, 2002], uses a full quantum QkNN approach based on the idea of the Hamming distance, a metric of similarity between binary vectors.

Similar to the other QkNN papers discussed, Trugenberger provides a mathematical approach for their QkNN operations but does not provide any circuit illustrations for their implementation. It does however discuss the quantum speedup advantage of the Hamming distance QkNN approach, with a focus on the proposed storage advantage when using larger datasets, as compared to classical kNN. As explained in Section 2.3.1, the Hamming distance encodes all distances between the input and the entire training set in one run. As noted by [Schuld et al., 2014b], the focus on storage optimisation is a different and interesting take on one of the positive outcomes of QkNN, and it will have advantageous implications as the size of data in our digital space continues to grow The author also notes that the *"power of this routine only becomes visible if it is applied to a large superposition of training states in the first register"* [Schuld et al., 2014b]. The results chapter in this dissertation in Section 5.3 illustrates that when QkNN is evaluated on a quantum system, it vastly outperforms kNN when applied to both a small and a larger set of data.

When detailing the current domain of QSVM, there is only one publication listed, [Rebentrost et al., 2014] *Quantum Pattern Recognition*. For NISQ (Noisy Intermediate-Scale Quantum) devices this QSVM approach is the primary implementation; however, quantum annealing devices [Dilmegani, 2021b] use binary classification for multi-classification problems. Binary classification models like SVM do not support multi-class classification natively. One way of dealing with this is by splitting the multi-class

classification dataset into multiple binary classification datasets and then using a binary classification model on each [DEMA et al., 2020].

Rebentrost et al.'s QSVM approach is a quantum enhanced approach, and it shows that a Quantum Support Vector Mechanism can be implemented with $O(logNM)$ run time in both the training and classification stages. As noted by Schuld et al., the work by Rebentrost et al. assumes an oracle for data encoding. It does not describe the encoding process either mathematically or through the use of circuit illustrations. From the oracle, quantum vectors are returned and evaluated using the inner product evaluation. This QSVM approach is a quantum enhanced approach, as these quantum vectors are then prepared for a classical kernel matrix. (This approach is used in the present work in Section 4.2.2.) Schuld et al. note that *"Rebentrost, Mohseni and Lloyd claim that in general, the evaluation of an inner product can be done faster on a quantum computer"* [Schuld et al., 2014b]. While this claim is explored in the results chapter in Section 5.3 – which shows that overall this assertion holds true – Rebentrost et al. do not make use of a quantum system to evaluate their claim.

[Schuld et al., 2014b] *An introduction to quantum machine learning* is an introductory step into the different avenues of quantum computing. It details the current work and approaches that have been investigated and their strengths, while also detailing areas that require clarification through re-implementation. The authors found that when exploring quantum machine learning there are two approaches that are taken in different research papers:

1. *"Many authors try to find quantum algorithms that can take the place of classical machine learning algorithms to solve a problem, and show how an improvement in terms of complexity can be gained."*
2. Others use the probabilistic description of quantum theory in order to describe stochastic processes. This is seen in the quantum hidden Markov models and the Bayesian theory, which helps to generalise these models.

The first point is predominantly true for k-Nearest Neighbour, kernel and clustering methods, in which expensive classical distance calculations are replaced by lower cost quantum computations. The subsequent research explored in [Schuld et al., 2014b]'s paper, provides mathematical designs for these quantum algorithms and quantum logic. The authors do not use quantum circuit implementations to explore and validate their quantum advantage claims. This dissertation provides implementations for the quantum algorithms, and demonstrates how the quantum data encoding is accomplished. It also evaluates these circuits' quantum advantages using a quantum computer.

3.2 A Focus on Implementation

In this section, three papers are considered from the perspective of implementation: a paper by Sharma, another paper by Kok and the Qiskit Tutorial [Qiskit, IBM Quantum Experience, 2020a]. They contribute more implementation details to the quantum algorithms detailed in the publications in the groundworks section.

The implementations outlined in these works are taken up later in the dissertation, where they are more fully elaborated and evaluated in a modular structure.

3.2.1 Sharma: Using QEML to implement a kNN Algorithm

[Sharma, 2020]'s *QEML: (Quantum Enhanced Machine Learning) Using Quantum Computation to implement a K-nearest Neighbors Algorithm in a Quantum Feature Space on Superconducting Processors*, provides *"a side-by-side comparison of both classic ML and QEML"* and makes use of *"tests to compare both the traditional KNN to its quantum variant (QKNN)."* Along with the detailed exploration of QkNN, this research paper provides detailed background and circuit illustrations for Grover's Search algorithm, as well as the necessary information for a QSVM implementation.

When detailing QSVM, Sharma provides an analysis into how one could implement a QSVM circuit, along with an illustration of a possible circuit implementation. When explaining Grover's Search algorithm, a circuit diagram is provided. However, the diagram is not executed on a quantum machine or on a quantum simulator. Grover's algorithm is only detailed in order to illustrate the speed-up advantage that quantum computers can provide. Section 4 of this dissertation details implementations of both QSVM and Grover's Search algorithm, providing implemented modular circuits for both algorithms. The quantum speed up claims are evaluated using a quantum machine.

Sharma's work explores a QEML approach of QkNN using a concept known as *fidelity* (discussed in Chapter 2), as well as a full quantum machine learning approach based on the Hamming distance (also implemented in this dissertation). When explaining fidelity, Sharma provides the necessary background into why fidelity could be used instead of the Hamming distance, stating that it *"promotes both faster execution due to quantum algorithms and more storage capabilities due to an enhanced feature space in a quantum register"*. This detailed analysis is accompanied by a Qiskit circuit diagram. While Sharma explains why fidelity would be used, the paper chose to explore the full quantum QkNN approach using the Hamming distance, instead of the quantum enhanced approach of fidelity. When detailing the Hamming distance implementation, Sharma provides an overview of the intended circuit diagram and a circuit implementation that was built using the IBM Q experience.

At the time the paper was written, Sharma noted technological limitations as follows:

"physical quantum computers are not accessible to the wide population and are only accessible for researchers and collaborators at recognized institutions." These restrictions resulted in none of the circuits in Sharma's work being executed on a quantum computer. The Hamming distance QkNN circuit was instead executed on a quantum simulator. When evaluating the implemented circuits, the Wisconsin Breast Cancer dataset [Dr. William H. Wolberg, 2010] was used with a dimension of 10. While the research paper does not go into detail about the type of data encoding used, it would appear that each datapoint was mapped to a qubit. As the size of qubit circuits is currently limited to 15 qubits [1], this type of data encoding limits the number of data points that could be used.

While Sharma's work is a starting point for detailing multiple quantum algorithms in one central location, they do not implement nor provide details of data encoding, and as they were limited in their available resources, they could not provide quantum execution analysis. This dissertation builds on the work of Sharma by providing: QkNN, QSVM and Grover's Search algorithm implementations and two implementations of data encoding. These circuits are evaluated using a quantum simulator and a quantum machine. This dissertation also answers the question posed in Sharma's work: *"how is the performance difference between KNN and QKNN affected when there are n = 100 dimensions or n = 1000 dimensions? Answering these queries will require future analysis."* The aforementioned quantum circuits are evaluated using the same Wisconsin Breast Cancer dataset, extended to incorporate 10, 100, 150 and 500 datapoints.[2] As detailed in Chapter 5, the quantum circuits in this dissertation are also evaluated using the Iris dataset, taking 10, 100 and 150 datapoints.[3]

3.2.2 Kok: Building a Quantum kNN Classifier with Qiskit

In [Kok, 2021] *Building a Quantum kNN Classifier with Qiskit: theoretical gains put to practice*, D.J. Kok implements a version of QkNN using the Dot Product method. The Dot Product method is not a full *quantum* QkNN approach but rather is a hybrid quantum algorithm – the neighbours are sorted according to a classical method described by [Basheer et al., 2021] (see also Chapter 2). Kok does provide a quantum machine learning implementation that details the data encoding, the quantum circuit and quantum execution with detailed code implementations alongside the necessary background information. While the author details more than one quantum data encoding technique, only one is implemented. The research paper does not provide multiple quantum machine learning algorithms or data encoding approaches.

The author details data encoding, specifically digital or binary encoding (as explained in

[1] For institutional use, the current largest quantum machine available is 50 qubits.
[2] The Wisconsin Breast Cancer dataset contains a total 569 datapoints.
[3] The Iris data contains a total of 150 datapoints.

Chapter 2) and analog or amplitude encoding. The paper provides an implementation and detailed explanation of the latter. While implementing a QkNN, the author does not make use of a full quantum machine during classification; rather a classical classifier is used. This classifier reads the quantum data using an oracle, which is executed using a quantum simulator. While Kok uses multiple datasets to analyse the quantum circuit (the Hofmann (German Credit dataset) [Hofmann, 1994], the HEP [Hobson et al., 2016] and the Iris datasets [Fisher, 1936]) a quantum machine is not used to evaluate these circuits, a quantum simulator is used instead. (This dissertation provides two implemented modular data encoding techniques and three quantum algorithms, which are evaluated using both a simulator and a real quantum device.)

In the implementation, the author makes use of the built in Qiskit QkNN function or *'oracle'* to perform the QkNN hybrid classification. (This dissertation provides an implementation of this approach, as well as a full quantum QkNN circuit, both of which are provided in a modular implementation.)

Kok's paper is a more complete example of an implemented and explained approach to a full quantum circuit that encompasses quantum data encoding, a quantum algorithm and a circuit analysis using a quantum computer, all in a central place. This dissertation will expand on this and other works mentioned in this literature review chapter, by providing multiple modular implementations for the data encoding, quantum algorithms and circuit analysis.

3.2.3 IBM: Qiskit Machine Learning Tutorial

At the time of writing, Qiskit provides a central tutorial page for various quantum machine learning algorithms [Development Team, Qiskit Machine Learning, 2021]. It details: *Quantum Neural Networks*, *Neural Network Classifier & Regressor*, *Quantum Kernel Machine Learning* and *qGANS*[4] *for Loading Random Distributions* techniques, in a central location. The tutorial site provides the code implementations for these algorithms along with some detail about the implementation process. However, it does not thoroughly detail the data encoding used nor does it execute these circuits on a quantum machine in order to evaluate them further.

Taking the Quantum Kernel Machine Learning tutorial, the document notes the need for data encoding. Through an illustration, a feature mapping technique (ZZfeature mapping) is specified. However, it does not explain how a general feature map works, rather it depicts its use within the code and links to another document that illustrates the circuit diagram for a ZZfeature map. This subsequent linked document also does not explain the generalised form

[4] qGAN or Generative Adversarial Network are used to *"learn the data's underlying random distribution and to load it directly into a quantum state"* [Team, 2021a]

of a feature map or the necessary background into how a feature map encodes the data.

When investigating this algorithm, the kernel component is similar to the approach used for QSVM (Quantum Support Vector Mechanism). As previously explained, QSVM is a quantum enhanced approach and not a full quantum machine learning algorithm, as the kernel is a classical kernel with quantum encoded data inputs. The Quantum Kernel algorithm in the Qiskit document site makes use of quantum data encoding but again only details it in the same manner as above. Within the implementation code, the quantum algorithm is executed on a quantum simulator. The tutorial site does not provide any detail about the expected execution output or how to interpret the output from the quantum simulator or from a real quantum device.

Qiskit provides introductory quantum videos and tutorials [Qiskit, IBM Quantum Experience, 2020a] centered on how to build a basic quantum circuit. These learning tools briefly explain the quantum background needed [Qiskit, IBM Quantum Experience, 2020b], but focus more on the code implementation. This tutorial page could be seen as the next step for a quantum enthusiast – from Qiskit's basic and introductory quantum circuit tutorials to an actual quantum machine learning implementation. The tutorial provides coded implementations and some explanation for each component of a circuit: data encoding, quantum algorithm and the quantum machine evaluation. However, many steps in the circuit are described in summary form and in many cases there is a need for the algorithm provided to be linked to further explanatory resources. As such, this tutorial site is more of a leap than a next step.

Chapter 4 of this dissertation aims to deliver the necessary background needed to understand how build each component of a quantum machine learning circuit. Each component of the data encoding is explained in the background chapter, Chapter 2, and each implementation step is explored in Chapter 4. Within Chapter 4, different approaches for data encoding and for quantum machine learning algorithms are provided in a modular form. The dissertation also evaluates each circuit on a quantum simulator and on the IBM quantum computer and the resulting outputs are measured.

4 Implementation

This chapter examines the implemented solution that aims to produce a modular quantum machine learning tool. The components of this solution are separated into subsections within which the important functions are explained. These subdivisions are used with the intent to provide a further understanding of the logic used in the system, thus providing an in-depth understanding into the overall structure.

4.1 Data Encoding

With the knowledge gained about quantum gates and operations from Chapter 2 we can now observe them in use. However, the question of how to use classical data with a quantum circuit arises. There are a number of data encoding techniques available and this section explores *amplitude encoding* and *feature mapping*.

4.1.1 Amplitude Encoding

Amplitude encoding is a technique to encode classical information into amplitude of quantum states. It maps the coordinates of a vector into the values of the amplitudes of a quantum state. As explained in Section 2.2.1, this type of data encoding is very useful for smaller datasets with multiple features; however, it could increase resource requirements [Team, 2021b].

Data Preparation

Amplitude encoding requires the mapped data vector to be normalised and to have a power-of-two dimension. To achieve this, data preparation through the use of data normalisation and data padding is needed.

Taking the desired dataset, the required variables are chosen, padded if necessary and the data is then re-normalised. For example, the Iris dataset is composed of four variables labelled -1 or 1. Keeping the features in the two first columns, the features can be transformed using the labels 0 and 1. This data is then padded with constant values and

re-normalised to result in a unitary vector.

```
data = np.loadtxt("/Users/ezi/Desktop/Disseration/Documentation/iris_classes1and2_scaled.txt")
X = data[:, 0:4]
print("First X sample (original)  :", X[0])

# pad the vectors to size 2^2 with constant values
padding = 0.3 * np.ones((len(X), 1))
X_pad = np.c_[np.c_[X, padding], np.zeros((len(X), 1))]
print("First X sample (padded)    :", X_pad[0])

# normalize each input
normalization = np.sqrt(np.sum(X_pad ** 2, -1))
X_norm = (X_pad.T / normalization).T
print("First X sample (normalized):", X_norm[0])

# angles for state preparation are new features
features = np.array([get_angles(x) for x in X_norm])
print("First features sample      :", features[0])

Y = (data[:, -1] + 1) / 2
```

Figure 4.1: Amplitude Encoding: Data Preparation

Data Mapping

To encode the prepared data into a quantum state, the data must be rotated in the Bloch sphere. The amplitude circuit makes use of the Ry gate to rotate single qubits through an angle around the $y - axis$. Given that the unitary vector has four dimensions, five angles with the function `get_angles()` shown in Figure 4.2 can be extracted; these angles then serve as arguments to encode the quantum circuit.

```
def get_angles(x):

    beta0 = 2 * np.arcsin(np.sqrt(x[1]) ** 2 / np.sqrt(x[0] ** 2 + x[1] ** 2 + 1e-12))
    beta1 = 2 * np.arcsin(np.sqrt(x[3]) ** 2 / np.sqrt(x[2] ** 2 + x[3] ** 2 + 1e-12))
    beta2 = 2 * np.arcsin(
        np.sqrt(x[2] ** 2 + x[3] ** 2) / np.sqrt(x[0] ** 2 + x[1] ** 2 + x[2] ** 2 + x[3] ** 2)
    )

    return np.array([beta2, -beta1 / 2, beta1 / 2, -beta0 / 2, beta0 / 2])
```

Figure 4.2: Getting the Angles Code

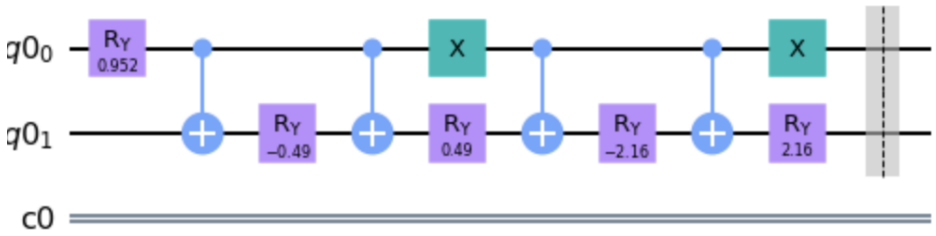

Figure 4.3: Amplitude Encoding: Circuit

4.1.2 Feature Mapping

Feature mapping reduces the amount the resources required to describe a large set of data. As described in Section 2.2.2, unlike amplitude encoding, the data points are converted into vectors using the $V(\phi(\overrightarrow{x_i}))$ function as quantum data is represented as vectors; this vector data can more easily encoded into a quantum space.

Data Preparation

Taking the Iris data set as our example in Figure 4.4, the number of datapoints one wishes to test needs to be specified. Similar to amplitude encoding, the desired data first needs to be normalised before it can be mapped.

```
iris = datasets.load_iris()
x = iris.data[:, :4]  # we only take the first two features.
Y = iris.target
```

```
#x, Y = load_iris().data, load_iris().target
x, Y = shuffle(x,Y)

# take the first 5
x = x[:5]
Y = Y[:5]
```

```
print(x,Y)
```

```
[[6.1 2.8 4.7 1.2]
 [5.7 3.8 1.7 0.3]
 [7.7 2.6 6.9 2.3]
 [6.  2.9 4.5 1.5]
 [6.8 2.8 4.8 1.4]] [1 0 2 1 1]
```

```
#noramlise the data
data = normalize(x)
print(data)
```

```
[[0.73659895 0.33811099 0.56754345 0.14490471]
 [0.8068282  0.53788547 0.24063297 0.04246464]
 [0.70600618 0.2383917  0.63265489 0.21088496]
 [0.73350949 0.35452959 0.55013212 0.18337737]
 [0.76467269 0.31486523 0.53976896 0.15743261]]
```

```
featuremap_circ = ZFeatureMap(4, reps=1)
```

Figure 4.4: Data Preparation

Data Mapping

There are different types of feature maps available:

1. `PauliFeatureMap`: The PauliFeatureMap is the general form and it allows for the creation of feature maps using different gates.
2. `ZFeatureMap`: The ZFeatureMap is the first-order evolution of the PauliFeatureMap.
3. `ZZFeatureMap`: The ZZFeatureMap is the second-order evolution of the PauliFeatureMap.

The Qiskit built in command, shown in Figure 4.5, allows any of these feature maps to be selected.

```
from qiskit.circuit.library import ZZFeatureMap, ZFeatureMap, PauliFeatureMap
```

Figure 4.5: Feature Map Import Code

The feature map is then called as a dataframe, specifying the number of features and the desired number of repetitions. Figure 4.6 details an implementation with four features and one repetition.

```
featuremap_circ = ZFeatureMap(4, reps=1)
```

Figure 4.6: Specifying the features and repetitions for the Feature Map

The data is then assigned to the circuit parameters, after which the circuit containing the feature map is combined with the assigned parameters of the second datapoint. Using the Qiskit code in Figure 4.7, the previous two points can be looped through to ensure the feature mapping is applied to every data point in the dataset.

```
circ = QuantumCircuit(4)
for i in range(len(data)):
    circ = circ.combine(featuremap_circ.assign_parameters(data[i]/2))
```

Figure 4.7: Feature Map Loop Application

4.1.3 SWAP Gate

When looking at the QkNN implementation in Section 4.2.1, the data is stored in qubit one. When encoding the data using feature mapping, the encoded data is in qubits two and three. To reduce the circuit and ensure all data points are encoded into qubit one requires the use of a SWAP Gate.

As illustrated in Figure 2.8, SWAP gates allow for the movement of information around in a quantum circuit without the loss of information. The SWAP gate is implemented by calling the Qiskit `swap` function. This function ensures that all data stored in qubit three is now located in qubit two. This SWAP operation is then repeated with qubit two and qubit one, allowing for the complete transfer of data to qubit one.

```
circ.swap([2], [3])
circ.swap([1], [2])
```

Figure 4.8: SWAP Gate: Code

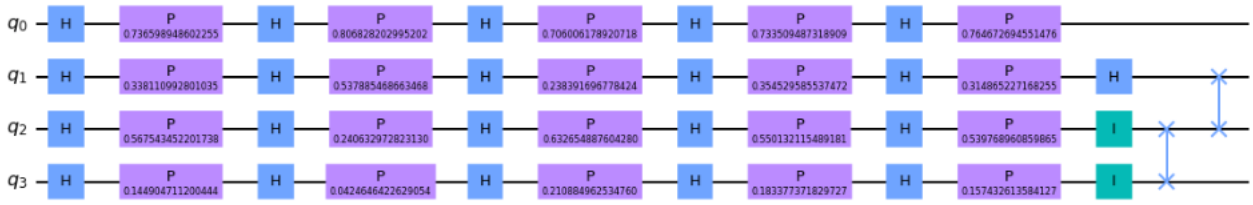

Figure 4.9: SWAP Gate: Circuit Application

4.2 Circuit Design

With the knowledge of how to encode classical data into a quantum state, the newly encoded quantum data can now be used by quantum machine learning algorithms. This section explores the implementation of pure quantum and quantum enhanced machine learning algorithms such as: Quantum k-Nearest Neighbour and Quantum Support Vector Mechanism, along with examining a quantum search algorithm, Grover's Search algorithm.

The Quantum k-Nearest Neighbour implementation is examined in detail in order to illustrate the use of quantum gates and to further understand their applications. The Quantum Support Vector Mechanism and Grover's Search algorithm are presented in a more shallow sense; however, two configuration options are provided for each.

4.2.1 Quantum k-Nearest Neighbour (QKNN)

The implemented QkNN algorithm follows the steps laid out in the work *Quantum Algorithm for k-Nearest Neighbours Classification Based on the Metric of Hamming Distance* [Wiebe et al., 2014].

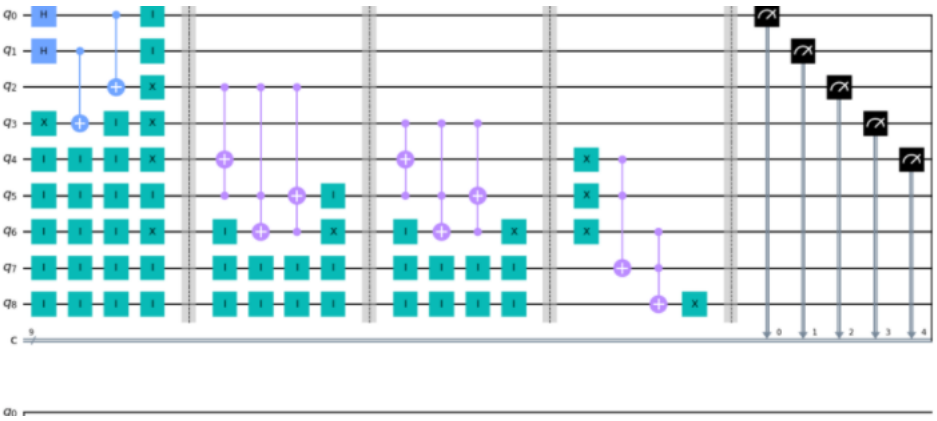

Figure 4.10: Completed Quantum K-Nearest Neighbour Circuit

This discussion will begin from step two of the main steps from Kaye [2004]. To provide an

overview of the operations, the qubits are first placed into superposition, after which two addition operations are applied to the circuit, followed by an OR operation, as seen in Figure 4.10. Each of the sections in this QkNN implementation are explored through the partitioned line brakes shown in Figure 4.10

Superposition Transformation

Figure 4.12 is a dissection of Figure 4.11, it shows the qubits that have the encoded data (qubit zero and qubit one) being placed into superposition, and they are then entangled with qubit two and qubit three respectively. As discussed in Section 2.4.2, this entanglement with other qubits protects *the information stored in qubits by not storing this information in individual qubits but rather in patterns of entanglement among many qubits.* This is a form of Quantum Error Correction, which protects the circuits *quantum information from errors that are primarily due to decoherence and quantum noise.*

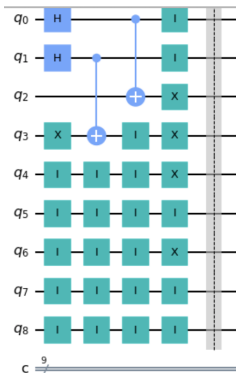

Figure 4.11: Quantum K-Nearest Neighbour Circuit: Superposition

Figure 4.12 displays the unclassified quantum state in qubit zero along with qubit one representing the training set. In order to encode both registers into superposition a Hadamard (H) gate is applied.

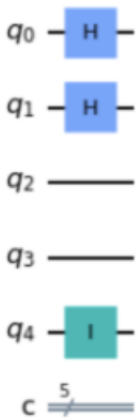

Figure 4.12: Quantum K-Nearest Neighbour Circuit: Superposition qubit zero and one

The states of qubit zero and qubit one are then entangled with qubits two and three, this allows for the superposition states to be mirrored in qubit two and three, respectively. This is implemented using a controlled not operation (CNOT) as demonstrated in Figure 4.13. In regard to Figure 4.13, an identity gate is observed on qubit four, this gate ensures no operation is applied to that qubit for one unit gate of time.

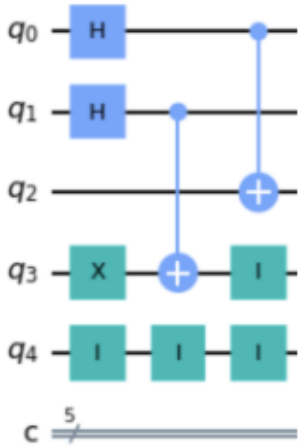

Figure 4.13: Qubit mirroring: Circuit

```
circuit.id(qreg_q[4])

circuit.h(qreg_q[0])
circuit.h(qreg_q[1])
```

Figure 4.14: Qubit mirroring: Code

Addition Operations

Figure 4.15 implements an initial A+D addition, using the entangled data in qubit two and three labelled as D0 and D1 respectively.

Taking qubit four as A0 and qubit five as A1, the following addition is preformed so as to satisfy the complete addition operation:

$$A0 + A1 = D0$$

$$A0 + A1 = D1$$

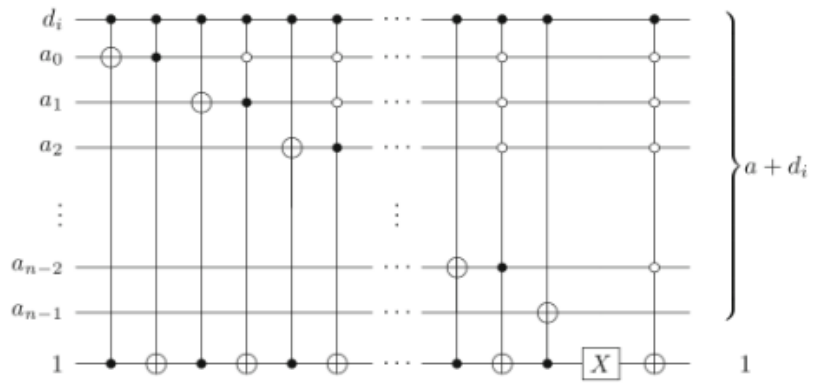

Figure 5

Figure 4.15: Addition operation illustrated in [Kaye, 2004]

A toffoli gate is used to carry out these additions, with the first addition operation being implemented following these steps:

1. Taking qubit two, qubit four is set as the target and qubit five is the control.

2. Then with qubit two and five as the controls, qubit six is the target.

3. Lastly, qubit five is the target with qubit six and two set as the controls.

After these operations, qubit six is negated, allowing the overflow to be retrieved. This overflow enable one to check if $D <= t$, where D is the first bit e.g 0010 and t is the comparison data bit 1100. The total Hamming distance is increased when the equation holds true.

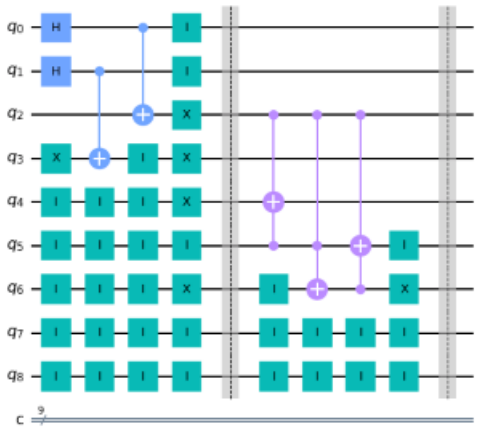

Figure 4.16: First Addition: circuit

The procedures carried out for the first addition, are then repeated with qubit three in place of qubit two. The code snippet in Figure 4.18 shows the resulting circuit with both

additions.

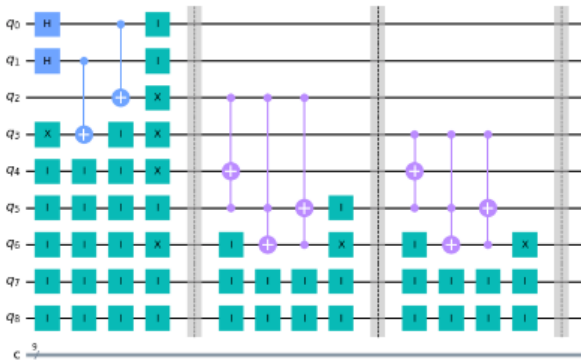

Figure 4.17: Circuit with Both Additions

```
circuit.x(qreg_q[3])#The NOT gat
circuit.cx(qreg_q[1], qreg_q[3])

circuit.id(qreg_q[4])
```

Figure 4.18: Second Addition: code

OR Operation

Following the addition operations, the condition of the Hamming distance can now be found. This is done through the application of an OR operation on the most significant bit. Keeping with the same paper [Kaye, 2004], the illustration in Figure 4.19 is followed in order to observe the Hamming distance.

If a *one* is found ,this is added to the Hamming Distance total and those with the same Hamming distance are said to be neighbours.

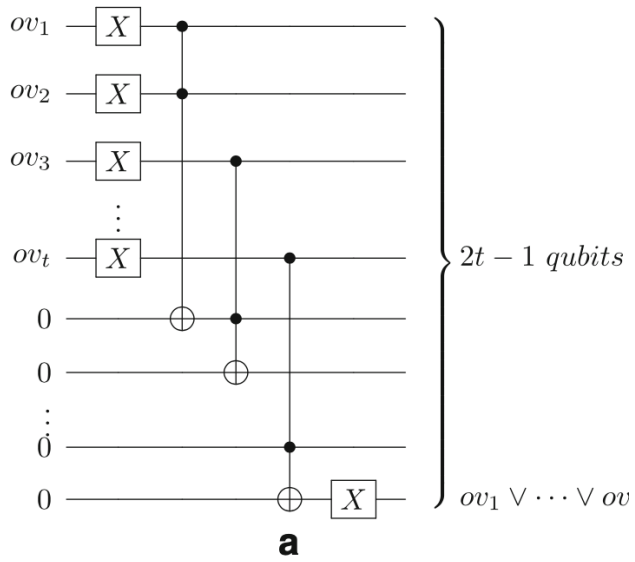

Figure 4.19: OR operation: [Kaye, 2004]

To begin with, all inputs from the additions stored in qubit four, five and six are negated. This negation allows for a zero or a '1' to be more easily detectable, with the presence of the latter depicting a neighbour.

To do so the following steps are applied:

1. Apply a toffoli gate on qubits four and five, with qubit seven as the target.

2. The second toffoli gate makes use of qubits five and six, with qubit eight acting as the target.

Following the application of both additions and the OR operation, the resulting complete QkNN circuit is seen in Figure 4.10.

4.2.2 Quantum Support Vector Mechanism (QSVM)

There are two possible directions that can be taken when implementing a Quantum Support Vector Mechanism:

1. Make use of the built in Qiskit function for QSVM.

2. Implement a QSVM circuit.

Function Call

Qiskit Aqua provides a predefined function that allows the entire QSVM kernel to be train. Figure 4.20 depicts how this function is imported and supplied with the necessary parameters:

- Feature map: This is a way of encoding classical data so it can be read by a quantum circuit.

- `Training_input`, `Test_input`: These are the data sets.

```
[ ]  from qiskit.aqua.algorithms import QSVM

     qsvm = QSVM(feature_map, training_input, test_input)
```

Figure 4.20: QSVM: Function Call

Circuitry

The implementation process for this QSVM technique can be divided into three parts:

1. **Data Processing:** Data processing is delivered through the use of feature maps as discussed in Section 4.1.2. Figure 4.21 shows how we can specify the feature map and the shot count for a dataset.

```
feature_map = ZZFeatureMap(feature_dimension=feature_dim, reps=2, entanglement='linear')
qsvm = QSVM(feature_map, training_input, test_input, datapoints[0])

backend = BasicAer.get_backend('qasm_simulator')
quantum_instance = QuantumInstance(backend, shots=100, seed_simulator=seed, seed_transpiler=seed)
```

Figure 4.21: QSVM: Feature Mapping

2. **Generation of the unitary operator:** Figure 4.22 shows the generation of the unitary operator, with a circuit composed of unitary gates and controlled not gates. The U gate is similar to the Ry gates, as it used for encoding classical data within the Bloch sphere. The U gates encodes this data by mapping the data around the $Z - axis$. *P.A McRae, M. Hilkea* [McRae and Hilke, 2020a] go more in depth into this implementation design, but with their direction the QSVM circuit can be implemented in Figure 4.22 .

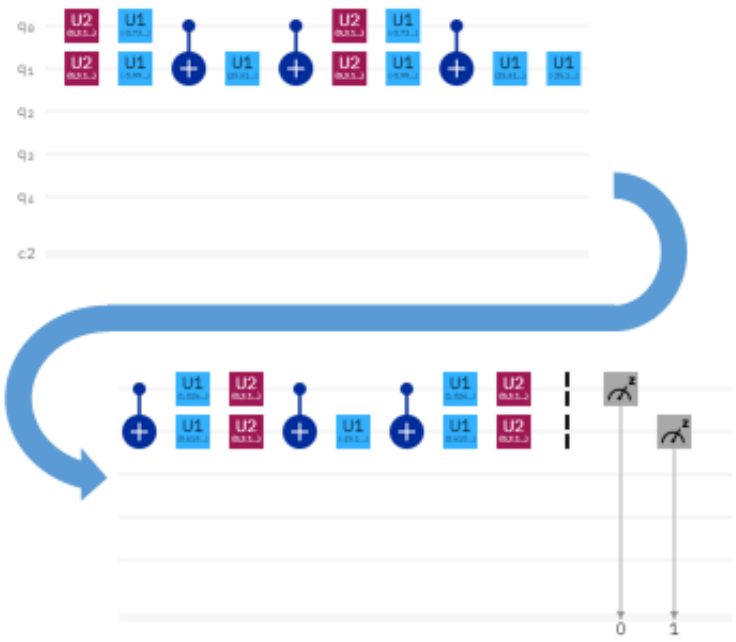

Figure 4.22: QSVM: Circuit McRae and Hilke [2020a]

3. **Generation of the kernel matrix K:** After the classical data is now encoded into quantum data it is supplied to the QSVM kernel. The kernel matrix in this implementation is the same as the classical SVM kernel but with quantum encoded data points. Using Python we can make use of the function call shown in Figure 4.23 to generate the kernel for our dataset.

```
kernel_matrix = result['kernel_matrix_training']
img = plt.imshow(np.asmatrix(kernel_matrix),interpolation='nearest',origin='upper',cmap='bone_r')
```

Figure 4.23: QSVM: Kernel Code

4.2.3 Grover's Search Algorithm

Similar to 2.1.2, two directions can be taken when implementing Grover's Search algorithm:

1. Make use of the built in Qiskit "black box" function.
2. Implement a Grover's Search circuit.

Function Call

Similar to QSVM, Qiskit Aqua provides a predefined function to perform Grover's Search algorithm. Using Qiskit we first call the import function, then the oracle. The oracle acts as

a black box reference to the Grover's Search circuit, which will only require the testing data as inputs.

```
from qiskit.aqua.algorithms import Grover

oracle = LogicalExpressionOracle(input_3sat)
grover = Grover(oracle)
```

Figure 4.24: Grover's Search Algorithm: Function Call

Circuitry

With Grover's Search algorithm, the circuit changes depending on the number of qubits and iterations needed. This implementation will be focusing on a four qubit circuit with one circuit iteration. Following the same guide as the QSVM circuit, we will implement another design from [McRae and Hilke, 2020a], where different rotations of Grover's Search with multiple iterations are more extensively covered.

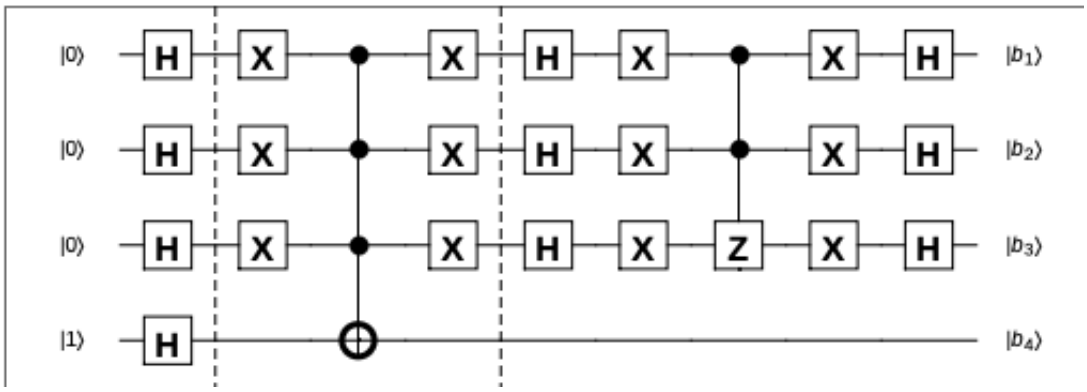

Figure 4.25: Grovers Search: Circuit McRae and Hilke [2020a]

Grover's Circuit is composed of three stages:

1. **Initialisation:** During the initialisation stage, all qubits are placed into superposition using the H gate.

   This ensures all the qubits have the same probability amplitude or have the same expected value.

2. **The Oracle:** The oracle is the black box function that the Qiskit function call initiates.

   The oracle function marks the state x' that satisfies the condition $f(x') = 1$ by performing a phase flip. All other states are left unaltered. The operation has the effect of inverting the state's amplitude and it runs in constant time.

This is implemented through marking the target element by negating its sign or applying a gate operation flip.

3. **Amplitude Amplification:** Amplitude Amplification amplifies the amplitudes of the desired element. It is applied to eliminate one-sided error from the inputs retrieved from the oracle.

   Applying these steps to a single qubit Grover's circuit would deliver Figure 4.26

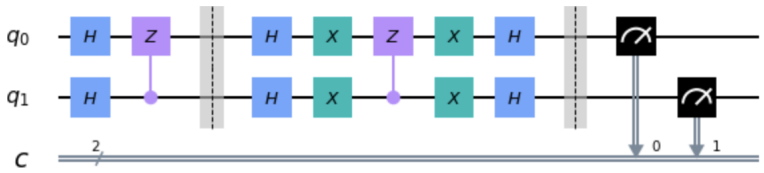

Figure 4.26: Grovers Search: 1 Qubit Implemented Circuit

These steps, for a full four qubit circuit would result in the circuit illustrated in Figure 4.27.

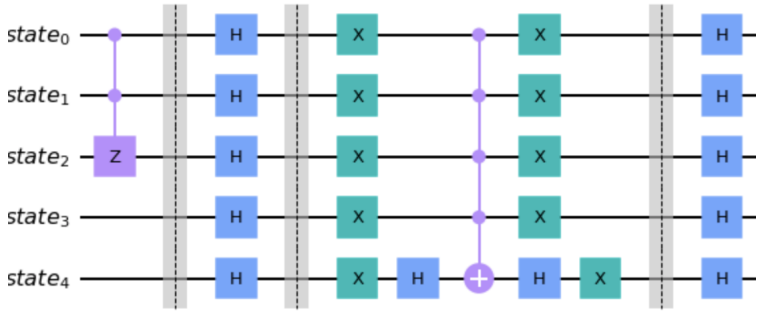

Figure 4.27: Grovers Search: Implemented Circuit

Since a four qubit Toffoli or Controlled Controlled Not (CCNOT) gate is not supported by Qiskit, it can be implemented using the Qiskit multi-qubit controlled not gate (MCMT), where the control qubits are specified first then the target qubit.:

```
MCMT.ccx(q[2], q[0], q[1], q[3], ctl1=None, ctl2=None, tgt=None).
```

Another option is to repeat the steps for the single qubit implementation above but repeated for each qubit operation as seen in Figure 4.28.

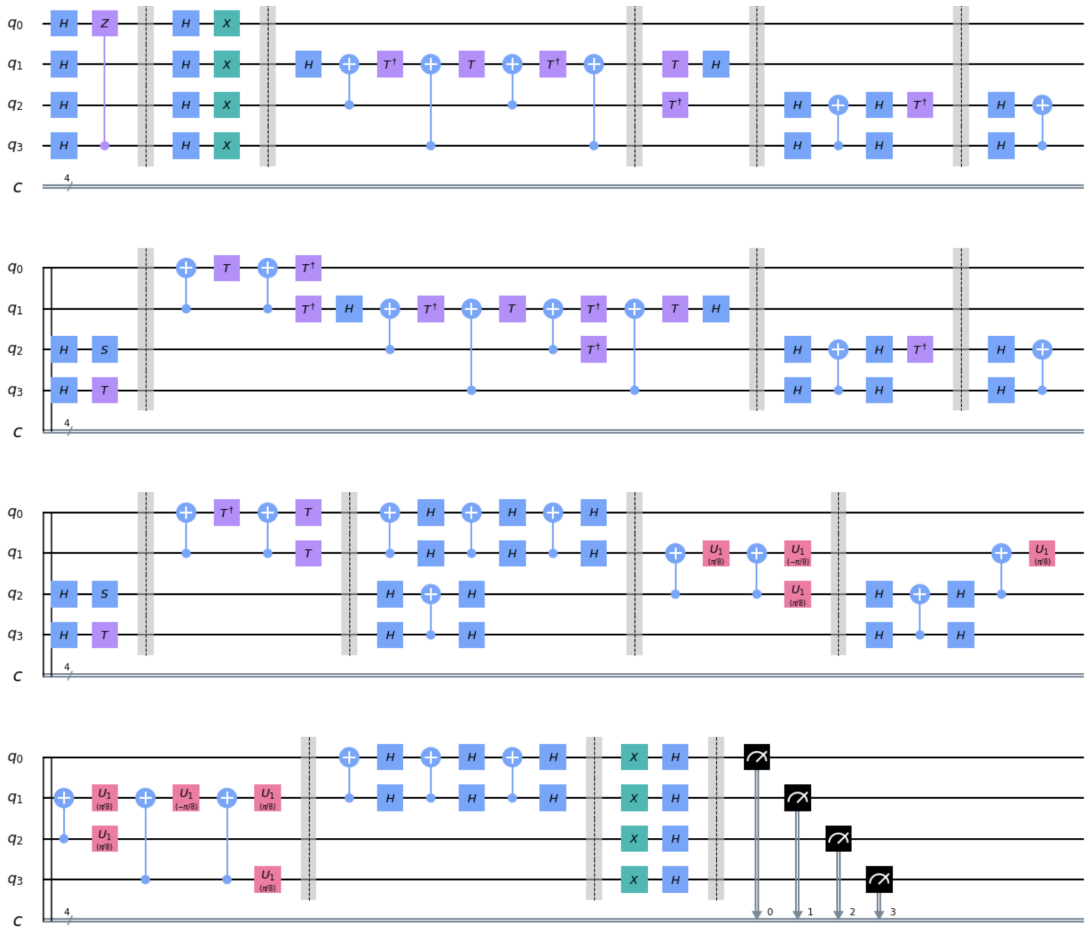

Figure 4.28: Grovers Search: Implemented 4 Bit Circuit

4.3 Measurements and Subroutines

To enable the circuit components in the quantum tool to be modular, quantum subroutines are used. This modular circuit can then be evaluated using a quantum machine. In order to evaluate the circuit output, a measurement must be taken.

4.3.1 Subroutines

Each algorithm and data encoding operation currently resides and are compiled in linear order in their code blocks e.g in a Jupyter notebook. These code blocks can be encapsulated as subroutines. Subroutines not only provide a visual representation of the circuit components but they also enable these components to be modular. Taking the feature mapping operation as an example, to be able to manipulate it as a subroutine one must:

1. Specify the number of qubits needed, in this case four.
2. Define the circuit by providing a name of the subroutine.

```
n = 4
FMap = QuantumCircuit(n,name='FMap')
```

Figure 4.29: Feature Map Encoding Subroutine: Steps 1 and 2

3. Carry out the feature mapping operation.
4. Set the feature map circuit to the subroutine as shown in Figure 4.30.
5. Call the `to_gate()` function in Figure 4.30, to enable the subroutine to be called and manipulated at any point in the circuit.

```
FMap= circ
print(FMap)

FMap.to_gate()
```

Figure 4.30: Encoding Feature Map Subroutine: Steps 4 and 5

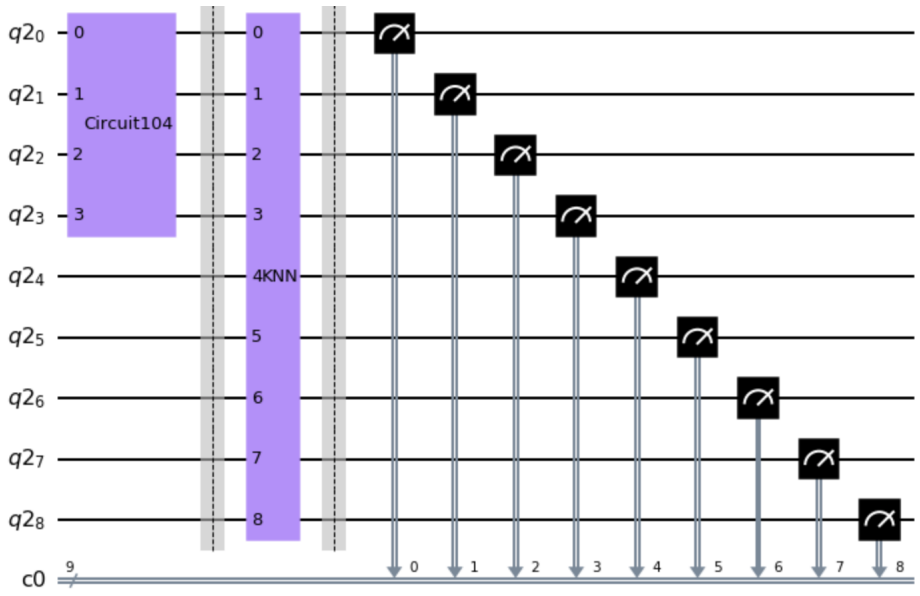

Figure 4.31: Subroutines: QKNN and Feature Mapping Circuit

The same process can also be applied to the QkNN circuit, this data encoding and quantum algorithm combination results in the circuit shown in Figure 4.31.

4.3.2 Measurements

At the end of Figure 4.31, there are an array of black time dials, these are measurements. In order to run the complete circuit the measurements of each qubit must be taken. This

measurement causes the superposition of the qubit to collapse and forces the qubits to reveal their state.

With the use of the Qiskit's measure function shown in Figure 4.32, the full circuit can be measured by supplying all used qubits. Individual qubits can also be tested during the circuit construction in order to check for a desired outcome.

```
circuit.measure([0,1,2,3,4,5,6,7,8], [0,1,2,3,4,5,6,7,8])
```

Figure 4.32: Taking Circuit Measurements

4.4 Quantum Execution

The completed circuits can now be run on a quantum computer however, there are a few preliminary steps that must be addressed. To begin with, an IBM Q Experience account is needed, so it must be created. Following the creation of the IBM account, the provided IBM access code needs to be noted. Finally with the IBM account, the access code can be loaded into the Qiskit code, as seen in Figure 4.33.

```
IBMQ.active_account()
```

```
IBMQ.save_account('b35ee002c40b58f68333b4dc663e57f118764a66394591bc1270dd258ede51076296505fe28203
f55935798b133fda9814350ca6b975b3899b38cbda9a034725', overwrite=True)
```

```
IBMQ.load_account()
provider= IBMQ.get_provider(hub='ibm-q')
real_device = provider.get_backend('ibmq_16_melbourne')
```

Figure 4.33: IBM Quantum Experience Setup

The Qiskit back-end that will run the circuit needs to specified. It is necessary to choose an appropriate back-end with the equivalent quantum volume availability. As seen in Figure 4.34, the maximum size of the implemented quantum circuit is the size of the quantum machine that must be chosen. With the largest implemented circuit (the QkNN circuit) using nine qubits, the Melbourne device is the only option as it allows for a 15 qubit volume and it is currently online.

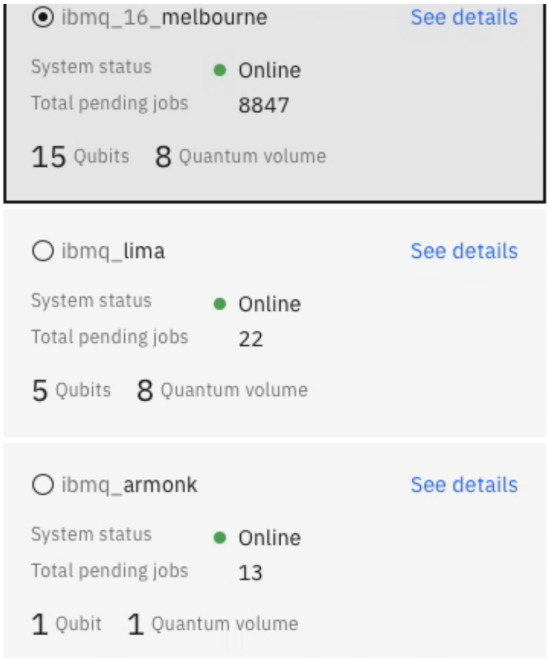

Figure 4.34: Quantum Machine Available

4.4.1 Quantum Simulation

Before, the circuit can be executed on a quantum machine, it first needs to simulated. This simulation ensures that one is not presented with an *'error running jobs'* alert, and thus the circuit is executable.

As shown in Figure 4.35, the simulator is initially called, then the simulation can be executed by providing the number of shots desired. As qubits in superposition are random, sometimes 0, sometimes 1, or a probability of both, shots allows for repeat measurements. This re-measurement determines the likelihood that a qubit is in a particular state. Following the simulation, the circuit results are then visualised.

```
from qiskit.visualization import plot_histogram
emulator = Aer.get_backend('qasm_simulator')

job = execute( circuit, emulator, shots=8192 )

hist = job.result().get_counts()
plot_histogram(hist)
```

Figure 4.35: Quantum Simulator: setting up the simulator for circuit execution and plotting the results

4.4.2 Quantum Run

Running a circuit on a quantum machine follows the same procedure as the quantum simulation. As seen in Figure 4.36, the only difference is the use of the quantum device (`backend = real_device`) that was initiated in Figure 4.33; the shot counts number and the return parameter `run_id` are also specified. The `run_id` parameter enables one to keep track of the current execution state, using this parameter the quantum run outputs can then plotted in order to visualise the result.

```
job = execute(circuit, backend = real_device, shots=8192)
print(job.job_id())
from qiskit.tools.monitor import job_monitor
job_monitor(job)
```

```
602f2ecd2717cee2b2d6969d
Job Status: job has successfully run
```

```
device_result = job.result()
plot_histogram(device_result.get_counts(circuit))
```

Figure 4.36: Quantum Machine Execution: running the circuit on a quantum computer

4.5 Classical Simulation of Quantum Circuits: JKU Implementation

The JKU simulator enables quantum circuits to executed on a simulation of a classical computer. Figure 4.37 depicts how to import and create an instance of the JKU simulator.

```
from qiskit_jku_provider import JKUProvider
#pip install qiskit-jku-provider
JKU = JKUProvider()
```

Figure 4.37: JKU Simulator: Import and Setup code

Using the code snippet in Figure 4.38, the JKU backend can be called from the JKU provider. This backend allows for the circuit to be simulated and then retrieved, from which the circuit results can be displayed.

```
# Get the JKU backend from the JKU provider
jku_backend = JKU.get_backend('qasm_simulator')

# Simulate the circuit with the JKU Simulator
job = execute(qc, backend=jku_backend)

# Retrieve and display the results
result = job.result()
print(result.get_counts(qc))
```

Figure 4.38: JKU Simulator: code to run JKU, execute the simulator and retrieve results

4.5.1 Error Handling

With Qiskit and classical simulators being relatively new, there are some errors that one will encounter. The open source community for JKU and Qiskit are active and available to help.

Coupling Map Error

One such error is the coupling map error, which is presented as a trace-back error when the JKU instance is initially called.

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-26-b9b2a69dcbd8> in <module>
----> 1 from qiskit_jku_provider import JKUProvider
      2 #pip install qiskit-jku-provider
      3 JKU = JKUProvider()

ModuleNotFoundError: No module named 'qiskit_jku_provider'
```

Figure 4.39: Expected error code output for Coupling Map error

Figure 4.40 illustrates the solution for this error, which requires one to locate the `quasm_simulator_jk.py` file from the location in the system where the JKU import was saved. When this file location is obtained one can then manoeuvre to the QasmSimulator class, there the `DEFAULT_CONFIGURATION` section can be found. Finally, in that `DEFAULT_CONFIGURATION` section the line, *coupling_map: None,* needs to be included.

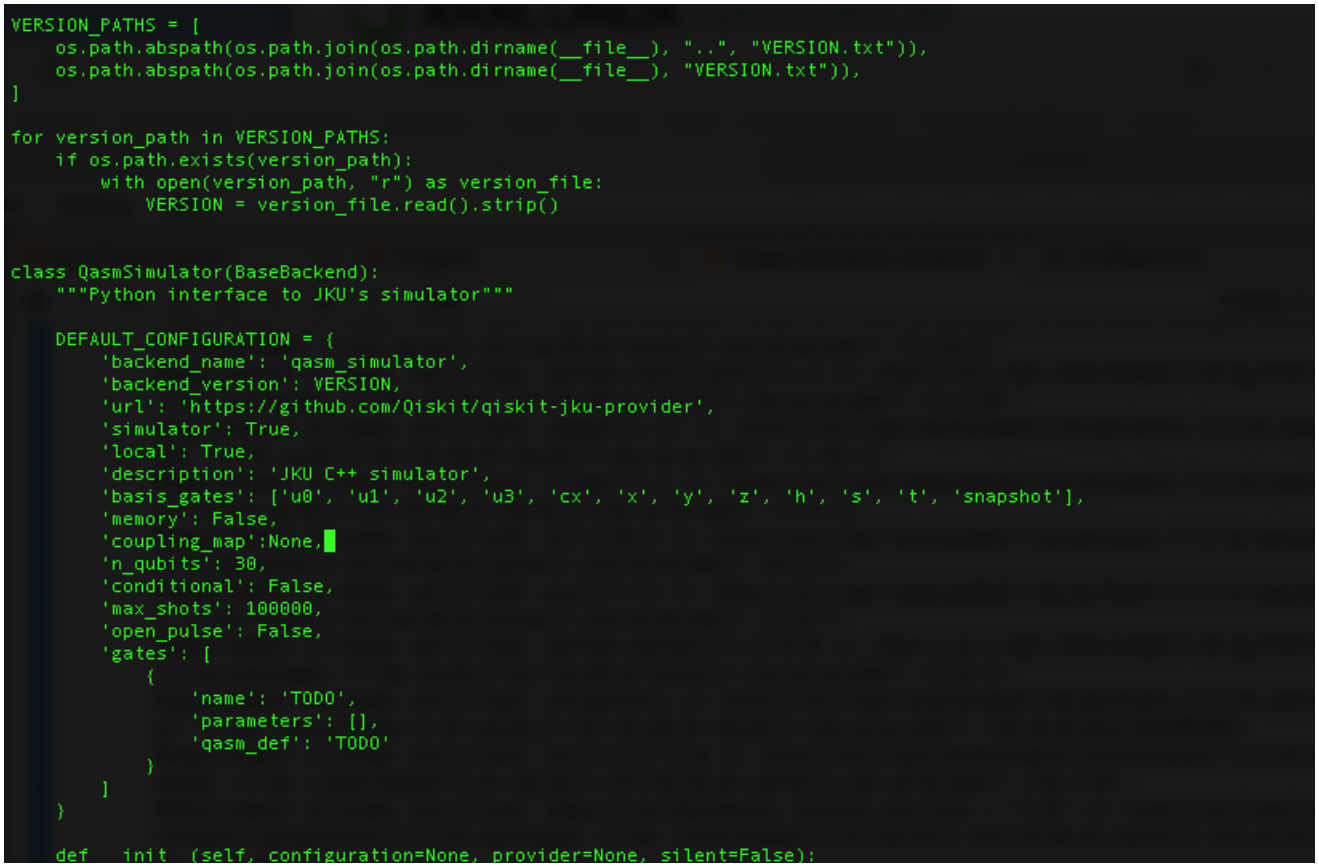

Figure 4.40: Coupling Map error: Error Code Fix

Identity Gate Error

The JKU simulator is built upon Qiskit's open source simulators. When Qiskit updates its simulators, these updates need to manually propagated to the JKU simulator. One such update is Qiskit backend simulator no longer automatically removing identity (ID) gates; *"IBMQ backends, for example, have traditionally implemented id gates as an implicit delay, though this behaviour will be removed in the future."* [Krsulich, 2021]

One possible fix would see the removal of the ID gates from ones circuit, if that does not fix the error – such was the case in this dissertation – one has to wait or can contribute to this update propagation within the JKU open source library.

5 Results

This section will present the results of the circuits implemented in Chapter 4. We will examine both quantum simulator and quantum machine outputs and the effects of noise and interference on these outputs. We will be observing the QkNN circuit with the Wisconsin Breast Cancer data set [Dr. William H. Wolberg, 2010] and the Iris dataset [Fisher, 1936]. For the Quantum Support Vector Mechanism and Grover's 'search algorithm, we will observe their results using the Wisconsin Breast Cancer data set and the 3-SAT [Conference, 2009] problem, respectively.

We will then observe the influence of shot increase in relation to accuracy. Finally, we will inspect the runtime of quantum machine learning algorithms in comparison to their classical counterparts.

5.1 Understanding the Results

As discussed in Section 2.4.2, errors, noise and interference have great effects on the outcome of a quantum process. This especially holds true with non simplistic circuits or circuits with greater depths. While we will not be observing the accuracy and the runtime of Grover's Search algorithm, we can observe the quantum simulation results using the 3-SAT input in Figure 5.1.

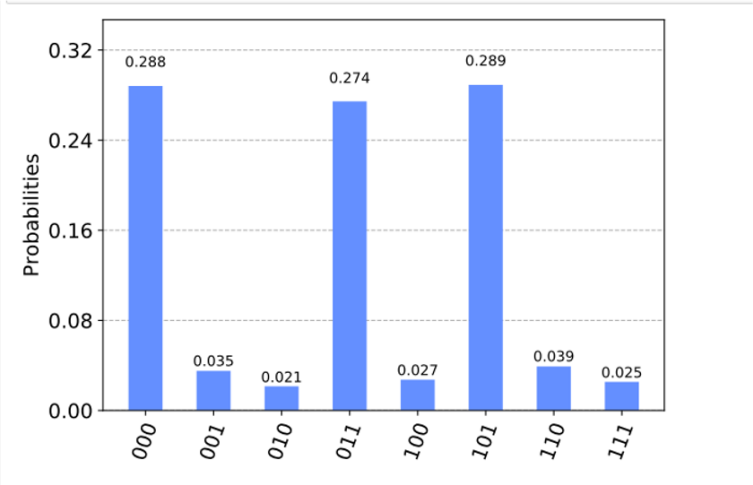

Figure 5.1: Grover's Search Algorithm results: 3-SAT Problem

Both Figure 5.1 and Figure 5.2 are noise free quantum simulations, with the former depicting Grover's Search algorithm on the 3-SAT input and the latter depicting the Quantum k-Nearest Neighbour circuit applied to the Iris data set.

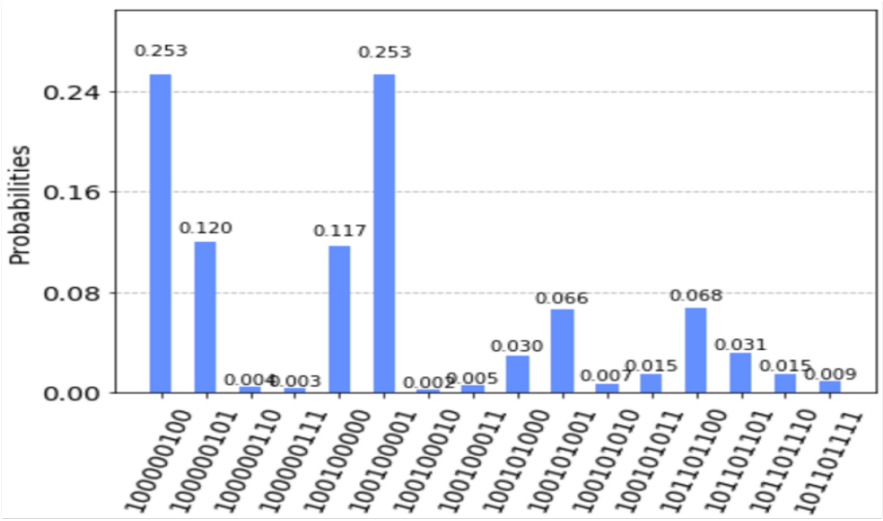

Figure 5.2: QkNN simulation results: Iris Dataset

Figure 5.3 illustrates the circuit result using the same datasets on a (real) 15 qubit quantum computer. The bottom axis depicts all possible results and the y-axis details the probability of finding those results within the circuit. Those with similar probabilities can be grouped together and are 'neighbours'. These probability groupings are easier to read and understand when viewing the quantum simulator output found in Figure 5.2, than using the quantum machine output shown in Figure 5.3. This is due to the output visualisation in Figure 5.3 encapsulating the noise, errors or gate inferences that may be present in our quantum circuit.

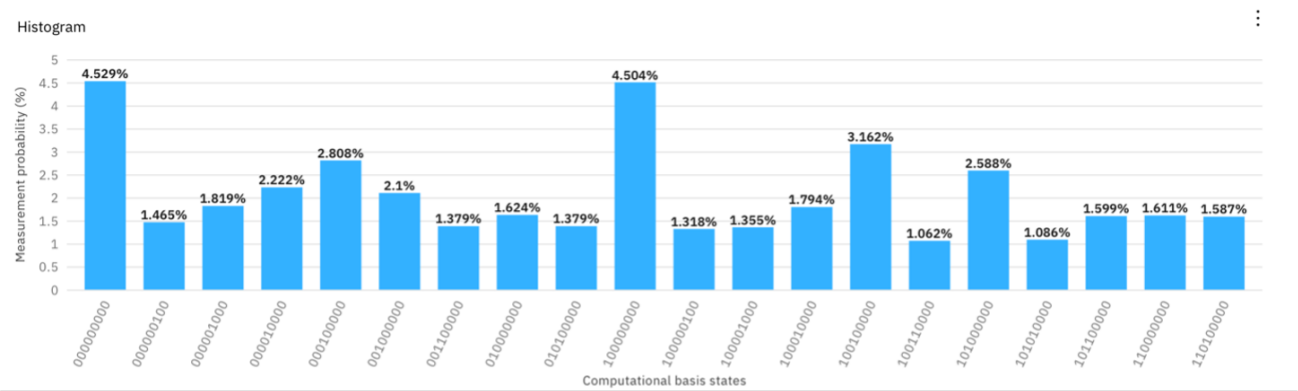

Figure 5.3: QkNN Quantum Run results: Iris Dataset

5.2 Shots and Accuracy

In Section 2.4.2 the use of shots in a quantum environment and its ability to reduce noise, interference and errors was discussed. While the reduction of these errors would theoretically increase an algorithm's accuracy, an increase in the number of shots does not necessarily retain in an increase in accuracy. Rather, it provides a more precise probability. This is because of errors such as gate interference, measurement errors, environmental noise and more, still exist within the device [Team, 2021c].

Table 5.1: Quantum Machine Learning algorithms accuracy vs shots

No. of Shots	QKNN Iris (%)	QKNN Breast Cancer (%)	kNN Iris (%)	kNN Breast Cancer (%)	QSVM Breast Cancer (%)	SVM Breast Cancer (%)
1	-	-	95	92	-	88
10	80	70	-	-	72	-
100	90	70	-	-	80	-
1000	95	80	-	-	87	-
8192	95	80	-	-	89	-

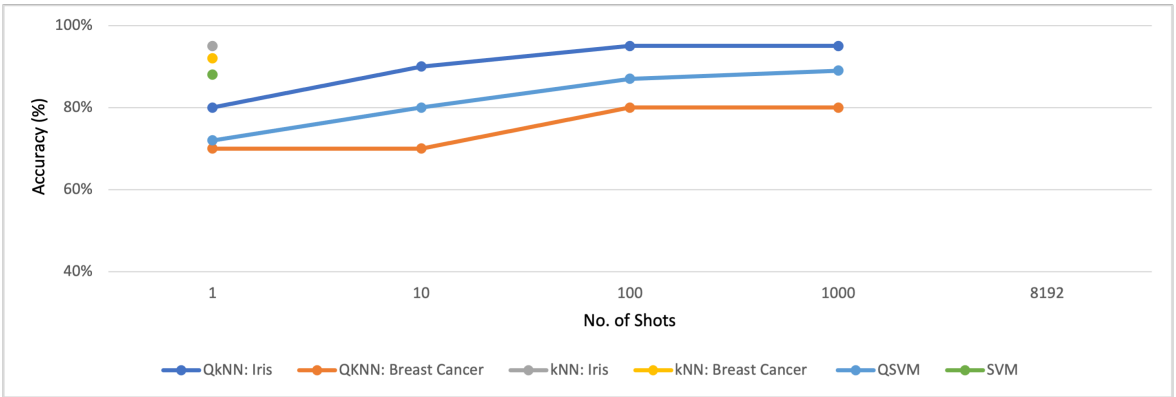

Figure 5.4: Quantum Machine Learning algorithms accuracy vs shots: Wisconsin Breast cancer and Iris datasets

In Figure 5.4, the resulting accuracy of each algorithm for the specified shot count may be seen. Of note, the dimensions are set to 100 for each dataset. The classical implementations were executed in a single shot, so they are represented as single dots in Figure 5.4 and their multi-shot inputs are omitted from Table 5.4. While an increase in the accuracy is seen, the quantum implementations do not surpass their classical counterparts. This is expected and can be attributed to various types of noise present in the quantum machine.

5.2.1 Shots and Runtime

As discussed, circuit executions are repeated in order to confirm the probability that a qubit is in the state measured. Each repetition is called a *shot*. With the circuit repeated multiple times, a question arises as to whether repetition affects circuit runtime. The Iris and the Breast Cancer datasets are used to evaluate this for the QSVM and QkNN circuits, with increasing shot counts of: 10, 100, 1,000 and 8,198 – with 8,198 being the maximum number of shots available for the quantum machines available[1]. For this evaluation, the number of datapoints taken was fixed at 100.

[1]The `quasi_simulator` or the quantum simulator in Qiskit has a maximum shot count of 65,535.

Table 5.2: kNN and QkNN Shots vs Runtime

No. of Shots	QkNN Iris (seconds)	QkNN Breast Cancer (seconds)	QSVM Breast Cancer (seconds)
10	10.6	10.9	92
100	12.2	11.2	107
1000	16.7	16.2	164
8192	52.2	52.5	878

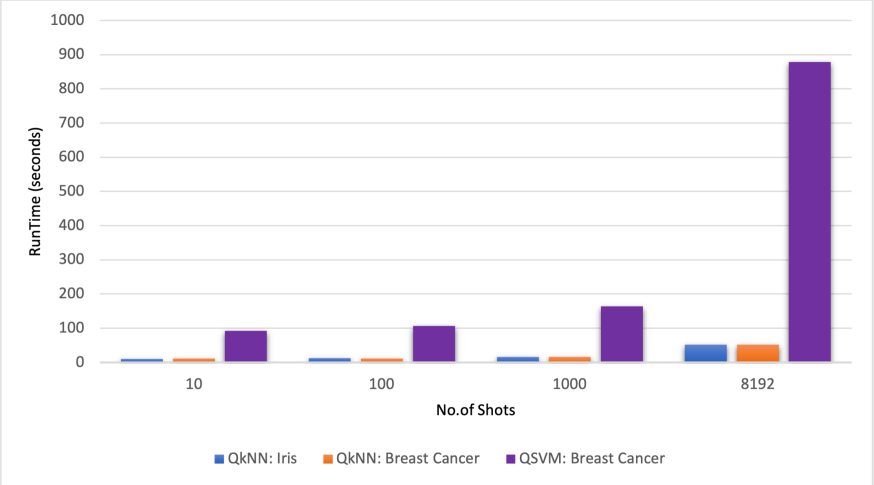

Figure 5.5: kNN and QkNN Shots vs Runtime

Figure 5.5 shows that an increase in the shot count causes an increase in the runtime.

5.3 Runtime

To truly test the proposed advantage of quantum computers, the execution time for both QkNN and QSVM in our datasets must to compared to their classical versions. We do so by increasing the data points or dimensionality of the input.

The Breast Cancer dataset and the Iris dataset used have maximum data points of 569 and 150 respectively. In Figure 5.6, we can see the runtime results for the QkNN circuit with the dimensions 10, 100, 150 and 500 data points for the Breast Cancer and Iris datasets. Figure 5.6 shows the clear quantum advantage of QkNN.

Table 5.3: kNN and QkNN Runtime vs Dimensions

Dimensions	QkNN Iris (minutes)	kNN Iris (minutes)	QkNN Breast Cancer (minutes)	kNN Breast Cancer (minutes)
10	1.1061	13.32	2.013	2.09
100	1.35	15.98	2.1545	3.97
150	1.512	23.48	2.398	5.89
500	1.763	-	2.525	19.83

Table 5.4: SVM and QSVM Runtime vs Dimensions

Dimensions	QSVM Breast Cancer (minutes)	SVM Breast Cancer (minutes)
10	3.2	3.17
50	4.58	4.5
150	4.57	5.89

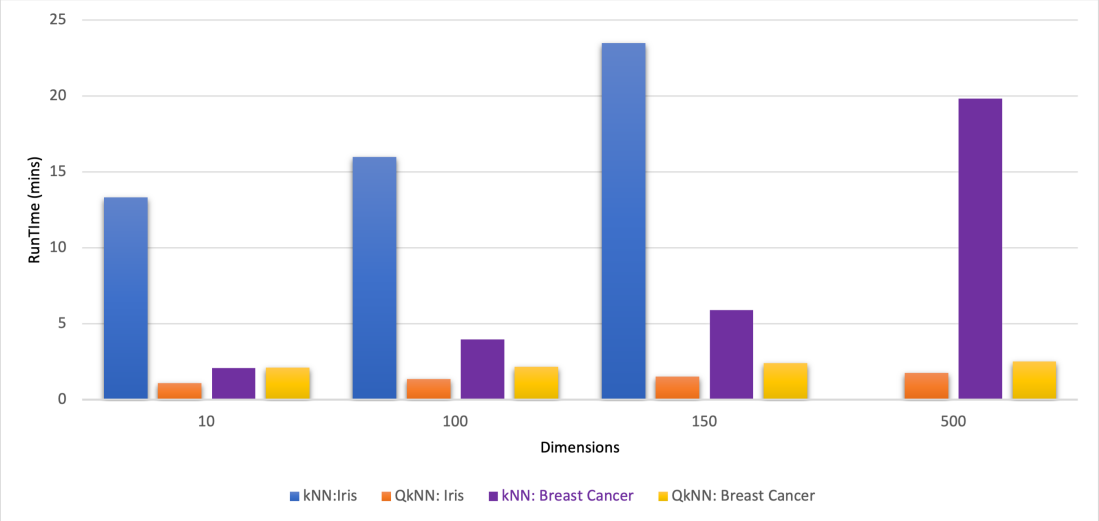

Figure 5.6: kNN and QkNN Runtime vs Dimensions: Wisconsin Breast cancer and Iris datasets

The QSVM dimensions for the Breast Cancer dataset were 10, 50 and 150. Figure 5.7 illustrates that the QSVM algorithm saw a little increase in speed. Overall it was surprisingly comparable to the SVM implementation.

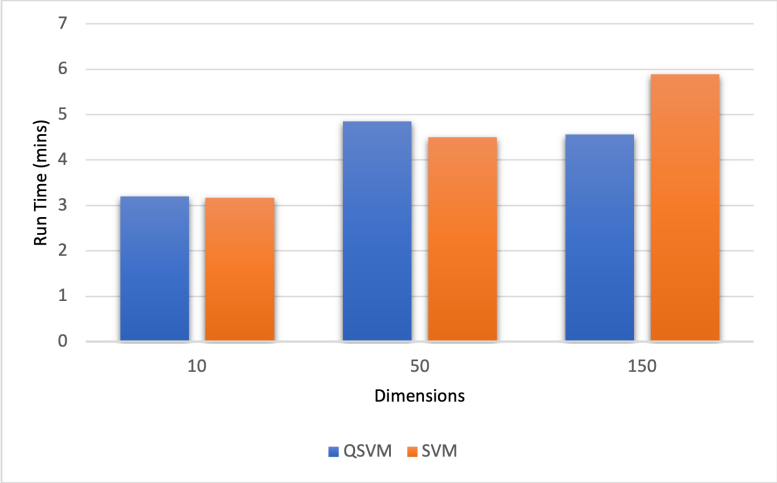

Figure 5.7: SVM and QSVM Runtime vs Dimensions: Wisconsin Breast cancer dataset

However, the QkNN circuit highlights the proposed advantage of quantum computers. We can see that the runtime results from QkNN shadow that of kNN. To note, when running a quantum system, the queue time can be long but the transplaining and execution time is mere seconds.

6 Evaluation

This dissertation has researched the application of cutting-edge quantum computing algorithms and data encoding techniques by translating their classically known methods to quantum algorithms, all encompassed in a modular tool.

Beginning with the quantum enhanced algorithm QSVM, this hybrid approach encompasses some classical variants of SVM, specifically within the kernel calculation. On the other hand, QkNN is a full quantum algorithm that is capable of classifying data by calculating and measuring the Hamming Distance between states. This chapter provides a summation of the results gained from implementing and executing both of these algorithms and the methods used to derive the outputs. Grover's Search algorithm is also considered.

6.1 Method

Before we evaluate our circuit outputs, we first consider other possible approaches for our quantum algorithms in comparison to the chosen implementations.

6.1.1 QSVM

In the implementation chapter we implemented [McRae and Hilke, 2020a] hybrid QSVM approach. An important critique of the QSVM implementation used in this work is that it is a hybrid instead of a pure quantum algorithm. This is due to the use of a classical kernel that reads quantum data inputs but executes the kernel classically. Another method using the D-Wave Quantum Annealer, described in [Willsch et al., 2020], is a true quantum SVM approach. In that work, an ensemble of different solutions was generated that often generalises better for unseen data than the single global minimum of an SVM trained on a conventional computer, especially in cases where only limited training data is available [Willsch et al., 2020]. Unfortunately, the choice to work with Microsoft's Qiskit's library for this dissertation and the lack of Quantum Annealers supported by Qiskit resulted in an inability to implement this QSVM approach.

6.1.2 QkNN

Other QkNN algorithms such as the Centroid classifier and the Dot Product were considered. The Centroid adaptation mentioned in [Lloyd et al., 2013] would see the kNN transformed into a k-means method, where each class is defined by the centroid of all data within that class. While this could bring an additional quantum algorithm to the modular tool, it has problems with information loss. If a class contains two distributed clusters, the centroid would point to the middle. This in turn would cause the classification to suffer.

The Dot Product method of [Basheer et al., 2020] is a hybrid quantum approach where the neighbours are sorted classically. While this approach would be interesting to contrast with, it does not provide the same benefits or advantages as the Hamming Distance QkNN, as it is a hybrid approach.

6.1.3 Grover's Search Algorithm

To continue with our quantum algorithms, the ability to test and compare a quantum centred algorithm [1] like Grover's Search algorithm in a classical environment would enable us to compare the methods and properties of this quantum implementation in a classical world. The JKU simulator enables quantum circuits' to be able to retain their quantum advantage, all while providing a fully comparative means of testing current classical machine learning algorithms within a quantum environment. This would be possible if the JKU simulator is updated to encompass full quantum based algorithms without error. To achieve this, continuous community collaboration is required in order to build up the JKU simulators and other quantum classical simulators.

6.2 Results

We can now evaluate our circuit outputs to see if they fulfilled quantum computing's claimed advantage, and where possible shortcomings could stem from.

6.2.1 Shot Count

When evaluating the quantum circuits with differing shot counts, Figure 5.5 showed that circuit repetitions cause the runtime to increase. There is marginal runtime change when increasing from shot count 10 to 100, with a more noticeable runtime increase between 100 and 1,000 and an even greater difference between 1,000 and 8,192. With the QSVM result, runtime is expected to be much higher than QkNN, due to the QSVM algorithm being a quantum hybrid algorithm with a classical kernel. However, Figure 5.6 showed that the

[1] By this is meant an algorithm that is not a quantum version of a classical machine learning algorithm.

performance of the QkNN algorithm outperformed the classical algorithm even with the highest shot count of 8,192.

The runtime of the QkNN algorithm with increasing shot count was evaluated using feature mapping for both datasets, as feature mapping reduces the resources required to describe a large set of data [Team, 2021b]. This evaluation could be repeated for amplitude encoding to observe any possible effect on the results. It could be anticipated that the runtime would be still quite low by comparison with classical kNN. Perhaps the difference in resource load for the two encoding types could be made more visible by increasing the size of the datasets. Unfortunately, this test could not be performed on the 15 qubit IBM quantum computer as it was removed from public access just before this hypothesis could be tested. The quantum circuits implemented in this work were nine qubits long. The current largest publicly available quantum machine accepts circuits with a maximum depth of five.

6.2.2 QSVM

Having considered the methods used for our quantum algorithms we now consider the results obtained. In relation to accuracy, both quantum machine learning algorithms did improve as the shot count increased. QSVM never surpassed the accuracy of the classical SVM, and in only one instance did it improve on the runtime of the classical SVM. The use of a hybrid QSVM approach, with the classical kernel, allows one to anticipate such a result.

6.2.3 QkNN

The QkNN with the Iris dataset almost reached the level of accuracy of the classical version. This result can be attributed to an increase in shot count, but also to the number of features chosen in relation to the size of the dataset. As will be clear from Figures 5.4 and 5.6, QkNN thrives on increased dimensionality. With a larger set of data, increased feature space and an improvement in noise reduction, QkNN could surpass the accuracy of the classical version. This would be dependent on an improvement in noise mitigation and quantum error reduction.

Increased dimensionality increases the run time of classical kNN, but QkNN does not suffer from this problem. Looking at Figure 5.6, all tested versions of QkNN outperformed the kNN implementations. This was not only expected but it also confirmed the capabilities of a quantum machine learning algorithm in comparison to its classical variant. As mentioned in Section 4.16, QkNN places the entire training set into superposition and this enables the algorithm to calculate all distances between the input and the entire training set in a single run. So one would only need as many qubits as are needed to encode the entire training set. This results in a much faster run time, as QkNN does not encounter the same bottleneck as kNN with increasing datasets.

7 Conclusion

In this dissertation, the quantum machine learning algorithms QkNN and QSVM were explored along with Grover's quantum based search algorithm and various data encoding techniques. The runtime advantages claimed for quantum systems were also evaluated.

We achieved the goal of implementing theoretical research and the theoretically based implementations of these algorithms and data encoding techniques. This work delivered them in a modular manner depicting the advantage not just of various quantum cloud computing providers but also the algorithms chosen and the encoding techniques available.

While building on the work of [Sharma, 2020], the number of tested dimensions was increased from 10 to 100, 150 and 500. With these increases, the run-time advantages claimed for QkNN and some variations of QSVM were observed in benchmarking.

While the implementation of the JKU classical quantum simulation was initiated, there is still more work to be done in order to see this implementation to completion. However, the main objective of providing an accessible and centralised pathway into quantum and quantum enhanced programming was achieved through the presentation of concise introductory information and the implementation of various modular components in an illustrative and accessible manner.

7.1 Further Studies

Looking ahead, the fields of quantum computing and quantum machine learning continue to innovate. While carrying out this dissertation we saw the release of a new quantum cloud system by Baidu Research called Paddle Quantum [Baidu, 2021]. Paddle Quantum is a quantum machine learning development toolkit that can help scientists and developers quickly build and train quantum neural network models and provide advanced quantum computing applications. Another example of continuing innovation is a new approach to quantum error correction from AWS Center for Quantum Computing that would encode the information into a protected qubit using many other physical qubits [Wheatley, 2021]. As

quantum computers are quite susceptible to noisy hardware and different types of errors and interference – as we discussed in Section 2.4.2 – this possible advancement could enable quantum computers to be able to *execute quantum algorithms despite the "noisy hardware" that remains prone to errors* [Wheatley, 2021]. This in turn could see our accuracy results improve significantly.

7.1.1 Modular Tool Improvements

In the scope of a modular tool, there is still more work to be done. To begin with, the addition of current quantum machine learning algorithms would expand on the modular tool's capabilities. Algorithms such as Quantum k-Means by Ullah Khan [Ullah Khan, 2019], for further grouping of new data points, would be generally useful in a modular system. Currently, to use this modular tool, one has to have the full code present and run all of it, calling the required modular components into the final circuit. It would be desirable to streamline this approach. One such implementation would see the tool being encapsulated in a library. This would allow one to call the desired modular components using the library's API calls without having the full code present.

As this work was coming to a close, IBM's 15 qubit quantum machine became unavailable for public use. The current largest publicly available IBM quantum computer allows for a circuit with a maximum depth of five. This change in circuit depth may not affect the current QSVM implementation as it is a hybrid implementation with only quantum encoded kernel data. The QkNN circuit implemented in this dissertation has a circuit depth of nine qubits and so, at the time of writing, can no longer be run on publicly available IBM quantum computers. Compression techniques, such as a reduction in the number of additions, could be implemented to reduce the number of qubits used. If IBM's quantum computer continues to only allow circuits with a maximum depth of five, a further exploration into the effect that compression could have on the QkNN circuit's ability to perform adequate classification could be undertaken.

7.1.2 Applications of Grover's Algorithm: Sawerwain and Wróblewski

During our discussion on Grover's Search algorithm in Section 2.3.3, we touched on its ability to have repetitions by increasing of the count size during re-measurement.

In the quantum recommendation system in Figure 7.1, the use of repetition would enable this system to be implemented. The addition of a *feature amplitude amplification* component would further allow for a quantum recommendation system to be viable, thus producing new analytical comparisons against the classical recommendation system application. More details of this solution are discussed in *Recommendation systems with the quantum k-NN and Grover algorithms for data processing* by Sawerwain and Wróblewski.

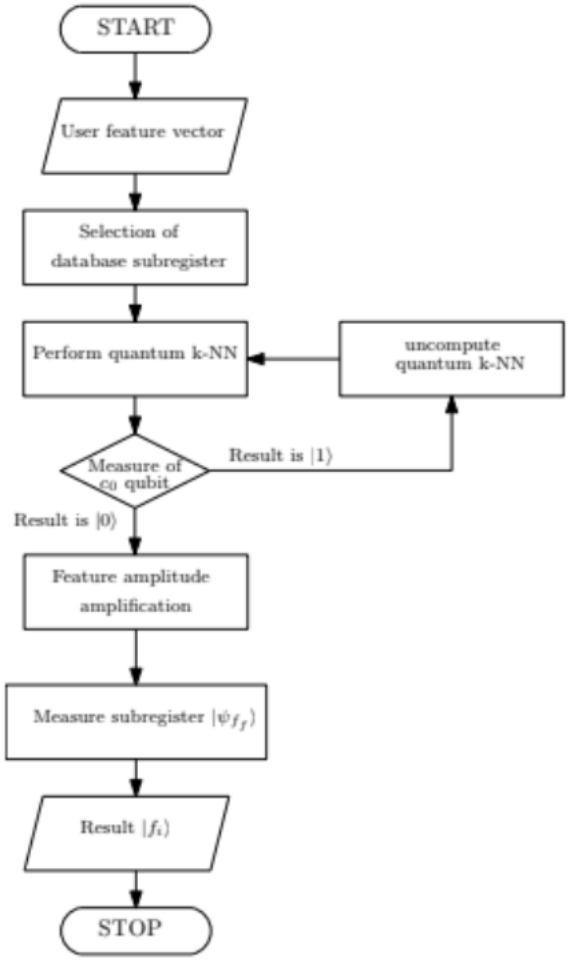

Figure 7.1: Control flow of proposed quantum recommendation system [Sawerwain and Wróblewski, 2019]

Continuing with Grover's Search, as it is a quantum based algorithm – not derived from a classical algorithm – there is no exact classical version we could compare it to. However, [Tomašić, 2020] compared Grover's Search algorithm to other classical search algorithms, namely Binary and Linear search. The paper found that Grover's Search algorithm did not outperform other search algorithms. However, the researcher executed Grover's search using a quantum simulator and not a real quantum device. Looking ahead, this research could be re-evaluated using the quantum machines that are currently available, in order to really compare Grover's quantum advantage to classical search.

7.1.3 JKU Simulator

As previously stated, the JKU simulator allows for quantum algorithms to be executed in a classical environment. It has been developed as an open source tool that is, unfortunately, currently producing errors. Further research and development into this or other similar systems is necessary. This would not only allow a circuit to keep its existing quantum advantages, but it would also facilitate a more comparative analysis between classical

machine learning algorithms, their quantum parts and also quantum centered algorithms such as Grover's Search.

7.1.4 Quantum Noise and Interference

Quantum noise and interference increases in proportion to circuit depth. There exist current error correction implementations that require multiple execution repetitions. While these techniques reduce some noise and errors, they are only applicable to smaller circuit designs. Another approach would involve increasing the current maximum number of qubits available in a quantum system. For example, if we wished to run Shor's algorithm [Fowler and Hollenberg, 2004] well enough to factor, say, a number 1,000 bits long – roughly the size used in some internet encryption schemes – the system will need to maintain logical qubits with a part-in-1-billion error rate [Cho, 2020a]. That may require entangling a grid of 1,000 physical qubits to safeguard a single logical qubit. By current research, this is a prospect that will take generations of bigger and better quantum computing chips [Cho, 2020b]. Quantum error correction and noise mitigation are critical issues, solutions to which would help propel quantum computing into the commercial arena [Wheatley, 2021]

It is still early days for quantum algorithms as they do not provide the same or a higher level of accuracy as classical methods. This, coupled with the resource-intensive and time-consuming nature of quantum computing queues [1], means that quantum machine learning algorithms are not yet suitable for general commercial use. As quantum computing and quantum machine learning continues to improve, it is hoped that these obstacles will diminish over time. We could see strides made to increase the scale of quantum computers to reduce resource load, noise reduction for greater consistency and an uptick in the open source community permitting more accessible and centralised theoretical knowledge, implementation know-how and high quality documentation.

As quantum machines grow in size and become more accessible, we can continue to build on work such as is presented in this dissertation and the research of others with the current technology at hand.

[1] While quantum computers execution times are reasonably fast, the queuing time to use a quantum machine can be more than three hours. The wait time depends on the number of active jobs waiting to be executed.

Bibliography

Marek Sawerwain and Marek Wróblewski. Recommendation systems with the quantum k-nn and grover algorithms for data processing. *International Journal of Applied Mathematics and Computer Science*, 29:139–150, 03 2019. doi: 10.2478/amcs-2019-0011.

Kasey Panetta. 5 trends emerge in the gartner hype cycle for emerging technologies, 2018, 2018.

Yisroel Meir Blumstein, Kalle Vedin, Tom Reding, Sammi Brie, Dr James V Stone, Peter Ells. Qubit, 2004. URL `https://en.wikipedia.org/wiki/Qubit`. [Online; accessed 26-April-2021].

The Jupyter Book Community. Single qubit gates, 2021. URL `https://qiskit.org/textbook/ch-states/single-qubit-gates.html`.

Sumsam Ullah Khan. Quantum k means algorithm. Master's thesis, KTH, School of Electrical Engineering and Computer Science (EECS), 2019.

John A. Cortese and T. Braje. Loading classical data into a quantum computer. *arXiv: Quantum Physics*, 2018.

Mohammad Jafarizadeh, Marziyeh Yahyavi, Ahmad Heshmati, and Naser Karimi. Quantum machine learning algorithms.
URL: `http://physics.sharif.edu/~qc/files/slides-2.pdf`, June 2020.

Phillip Kaye. Reversible addition circuit using one ancillary bit with application to quantum computing. *arXiv e-prints*, art. quant-ph/0408173, August 2004.

Paul-Aymeric McRae and Michael Hilke. Quantum-enhanced machine learning for covid-19 and anderson insulator predictions, 2020a.

Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, Oct 1997. ISSN 1095-7111. doi: 10.1137/s0097539795293172. URL `http://dx.doi.org/10.1137/S0097539795293172`.

Valentin Gebhart, Luca Pezzè, and Augusto Smerzi. Quantifying computational advantage of grover's algorithm with the trace speed. *Scientific Reports*, 11(1), Jan 2021. ISSN 2045-2322. doi: 10.1038/s41598-020-80153-z. URL http://dx.doi.org/10.1038/s41598-020-80153-z.

Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. An introduction to quantum machine learning. *Contemporary Physics*, 56(2):172–185, Oct 2014a. ISSN 1366-5812. doi: 10.1080/00107514.2014.964942. URL http://dx.doi.org/10.1080/00107514.2014.964942.

Alastair A. Abbott and Cristian S. Calude. Understanding the quantum computational speed-up via de-quantisation. *Electronic Proceedings in Theoretical Computer Science*, 26:1–12, Jun 2010. ISSN 2075-2180. doi: 10.4204/eptcs.26.1. URL http://dx.doi.org/10.4204/EPTCS.26.1.

William Coffeen Holton. Quantum Computer, June 2021. URL https://www.britannica.com/technology/quantum-computer. Accessed: June 9, 2021.

Christine C. Moran. Quintuple: A tool for introducing quantum computing into the classroom. *Frontiers in Physics*, 6:69, 2018. ISSN 2296-424X. doi: 10.3389/fphy.2018.00069. URL https://www.frontiersin.org/article/10.3389/fphy.2018.00069.

Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. An Introduction to Quantum Machine Learning. *Contemporary Physics*, 56(2):172–185, Oct 2014b. ISSN 1366-5812. doi: 10.1080/00107514.2014.964942. URL http://dx.doi.org/10.1080/00107514.2014.964942.

Ozlem Salehi, Zeki Seskir, and Ilknur Tepe. A computer science-oriented approach to introduce quantum computing to a new audience. *IEEE Transactions on Education*, page 1–8, 2021. ISSN 1557-9638. doi: 10.1109/te.2021.3078552. URL http://dx.doi.org/10.1109/TE.2021.3078552.

Weiwen Jiang, Jinjun Xiong, and Yiyu Shi. When machine learning meets quantum computers. *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, Jan 2021. doi: 10.1145/3394885.3431629. URL https://arxiv.org/pdf/2012.10360.pdf.

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng.

Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, Savannah, GA, November 2016. USENIX Association. ISBN 978-1-931971-33-1. URL https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi.

Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

Sébastien Marcel and Yann Rodriguez. Torchvision the machine-vision package of torch. In *Proceedings of the 18th ACM International Conference on Multimedia*, MM '10, page 1485–1488, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605589336. doi: 10.1145/1873951.1874254. URL https://doi.org/10.1145/1873951.1874254.

Kishor Bharti, Alba Cervera-Lierta, Thi Ha Kyaw, Tobias Haug, Sumner Alperin-Lea, Abhinav Anand, Matthias Degroote, Hermanni Heimonen, Jakob S. Kottmann, Tim Menke, Wai-Keong Mok, Sukin Sim, Leong-Chuan Kwek, and Alán Aspuru-Guzik. Noisy intermediate-scale quantum (nisq) algorithms, 2021.

M. Lichman. UCI machine learning repository, 2013. URL http://archive.ics.uci.edu/ml.

Daniel Koch, Laura Wessing, and Paul M. Alsing. Introduction to coding quantum algorithms: A tutorial series using qiskit, 2019.

D.J. Kok. Building a quantum knn classifier with qiskit: theoretical gains put to practice, 2021.

Siddharth Sharma. Qeml (quantum enhanced machine learning): Using quantum computing to enhance ml classifiers and feature spaces, 2020.

Colm Ó hÉigeartaigh. Technical manua, 2003. URL https://hego.redbrick.dcu.ie/technicalmanual/.

Noson S Yanofsky and Mirco A Mannucci. *Quantum computing for computer scientists*. Cambridge University Press, 2008.

M. Weigold, J. Barzen, F. Leymann, and M. Salm. Data encoding patterns for quantum algorithms. In *The Hillside Group (Hrsg): Proceedings of the 27th Conference on Pattern Languages of Programs (PLoP '20)*, (PLoP '20), 2021.

Frank Leymann and Johanna Barzen. The bitter truth about gate-based quantum algorithms in the nisq era. *Quantum Science and Technology*, 5(4):044007, Sep 2020. ISSN 2058-9565. doi: 10.1088/2058-9565/abae7d. URL http://dx.doi.org/10.1088/2058-9565/abae7d.

Rodney David. Qa2. explaining variational quantum classifiers. *Medium*, 2021. URL https://medium.com/swlh/qa2-explaining-variational-quantum-classifiers-b584c3bd7849.

Ryan LaRose and Brian Coyle. Robust data encodings for quantum classifiers. *Physical Review A*, 102(3), Sep 2020. ISSN 2469-9934. doi: 10.1103/physreva.102.032420. URL http://dx.doi.org/10.1103/PhysRevA.102.032420.

Slimane Thabet. Decomposing step by step a quantum machine learning algorithm. *Towards Data Science*, 2020. URL https://towardsdatascience.com/decomposing-step-by-step-a-quantum-machine-learning-algorithm-8adfa66aed7e.

Maria Schuld. Using quantum machine learning to analyze data in infinite-dimensional spaces. *Medium*, 2018. URL https://medium.com/xanaduai/analyzing-data-in-infinite-dimensional-spaces-4887717be3d2.

Onel Harrison. Machine learning basics with the k-nearest neighbors algorithm. *Towards Data Science*, 2018. URL https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761.

Afrad Basheer, A. Afham, and Sandeep K. Goyal. Quantum *k*-nearest neighbors algorithm. *arXiv e-prints*, art. arXiv:2003.09187, March 2020.

Yue Ruan, Xiling Xue, Heng Liu, Jianing Tan, and Xi Li. Quantum Algorithm for K-Nearest Neighbors Classification Based on the Metric of Hamming Distance. *International Journal of Theoretical Physics*, 56(11):3496–3507, November 2017. doi: 10.1007/s10773-017-3514-4.

PBerwick R. An idiot's guide to support vector machines (svms)., 2003.

Paul-Aymeric McRae and Michael Hilke. Quantum-enhanced machine learning for covid-19 and anderson insulator predictions, 2020b.

Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, page 144–152, New York, NY, USA, 1992. Association for Computing Machinery. ISBN 089791497X. doi: 10.1145/130385.130401. URL https://doi.org/10.1145/130385.130401.

Patrick Huembeli. Introduction into quantum support vector machines. *Medium*, 2019. URL https://medium.com/@patrick.huembeli/introduction-into-quantum-support-vector-machines-727f3ccfa2b4.

Srinivasan Arunachalam and Ronald de Wolf. Guest column: A survey of quantum learning theory. *SIGACT News*, 48(2):41–67, June 2017. ISSN 0163-5700. doi: 10.1145/3106700.3106710. URL https://doi.org/10.1145/3106700.3106710.

Frank Zickert. Towards understanding grover's search algorithm. *URL: https://towardsdatascience.com/towards-understanding-grovers-search-algorithm-2cdc4e885660*, March 2021.

Quantiki Team. Grover's search algorithm. *URL: https://www.quantiki.org/wiki/grovers-search-algorithm*, November 2015.

Eva Borbely. Grover search algorithm, 2007.

Cem Dilmegani. Cloud quantum computing & top cloud qc vendors in 2021. *URL: https://research.aimultiple.com/quantum-computing-cloud/*, December 2021a.

El C, ReyHahn, Le Deluge, . Hamiltonian (quantum mechanics, 2019. URL https://en.wikipedia.org/wiki/Hamiltonian_(quantum_mechanics). [Online; accessed 30-August-2021].

Cem Dilmegani. Quantum annealing in 2021: Practical quantum computing. *URL:https://research.aimultiple.com/quantum-annealing/*, January 2021b.

Rajdeep Kumar Nath, Himanshu Thapliyal, and Travis S. Humble. A review of machine learning classification using quantum annealing for real-world applications, 2021.

Ryan LaRose. Overview and comparison of gate level quantum software platforms. *Quantum*, 3:130, Mar 2019. ISSN 2521-327X. doi: 10.22331/q-2019-03-25-130. URL http://dx.doi.org/10.22331/q-2019-03-25-130.

Jack Krupansky. Shots and circuit repetitions: Developing the expectation value for results from a quantum computer. *Medium*, 2020. URL https://doi.org/10.1038/s41586-019-0980-2.

Dieter Suter and Gonzalo Alvarez. Colloquium: Protecting quantum information against environmental noise. *Review of Modern Physics*, 88:041001, 10 2016. doi: 10.1103/RevModPhys.88.041001. URL https://www.famaf.unc.edu.ar/~galvarez/2016-RMP-Suter_Alvarez-Protecting_QI.pdf.

Patrick J. Coles, S. Eidenbenz, S. Pakin, A. Adedoyin, John Ambrosiano, P. M. Anisimov, W. Casper, Gopinath Chennupati, Carleton Coffrin, Hristo Djidjev, D. Gunter, S. Karra, Nathan Lemons, Shizeng Lin, A. Lokhov, A. Malyzhenkov, D. Mascarenas, S. Mniszewski, B. Nadiga, D. O'Malley, D. Oyen, L. Prasad, Randy Roberts, P. Romero, N. Santhi, N. Sinitsyn, P. Swart, Marc Vuffray, J. Wendelberger, B. Yoon, R. Zamora, and W. Zhu. Quantum algorithm implementations for beginners. *ArXiv*, abs/1804.03719, 2018.

Wikipedia. Quantum error correction — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Quantum%20error%20correction&oldid=1022510317, 2021. [Online; accessed 20-May-2021].

Simon J Devitt, William J Munro, and Kae Nemoto. Quantum error correction for beginners. *Reports on Progress in Physics*, 76(7):076001, Jun 2013. ISSN 1361-6633. doi: 10.1088/0034-4885/76/7/076001. URL http://dx.doi.org/10.1088/0034-4885/76/7/076001.

Yehuda Naveh. Classical simulators for quantum computers, Feb 2019. URL https://medium.com/qiskit/classical-simulators-for-quantum-computers-4b994dad4fa2.

Esma Aïmeur, Gilles Brassard, and Sébastien Gambs. Machine learning in a quantum world. In *Conference of the Canadian Society for Computational Studies of Intelligence*, pages 431–442. Springer, 2006.

Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum algorithms for supervised and unsupervised machine learning, 2013.

Carlo A Trugenberger. Quantum pattern recognition. *Quantum Information Processing*, 1 (6):471–493, 2002.

Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. Quantum support vector machine for big data classification. *Physical Review Letters*, 113(13), Sep 2014. ISSN 1079-7114. doi: 10.1103/physrevlett.113.130503. URL http://dx.doi.org/10.1103/PhysRevLett.113.130503.

BA DEMA, Junya ARAI, and Keitarou HORIKAWA. Support vector machine for multiclass classification using quantum annealers, 2020.

Qiskit, IBM Quantum Experience. Qiskit algorithms - coding with qiskit season 2, 2020a. URL https://www.youtube.com/playlist?list=PLOFEBzvs-VvrhKYASly1BXo1AdPyoCsor.

Olvi L. Mangasarian Dr. William H. Wolberg, W. Nick Street. UCI machine learning repository, 2010. URL https://archive.ics.uci.edu/ml/datasets/iris.

Afrad Basheer, A. Afham, and Sandeep K. Goyal. Quantum *k*-nearest neighbors algorithm, 2021.

R Hans Hofmann. German credit dataset http://archive. ics. uci. edu/ml/datasets/statlog+. oa mohamed jafar and r. *Sivakumar R*, 1994.

Peter Hobson, Brian C Lovell, Gennaro Percannella, Alessia Saggese, Mario Vento, and Arnold Wiliem. Hep-2 staining pattern recognition at cell and specimen levels: datasets, algorithms and results. *Pattern Recognition Letters*, 82:12–22, 2016.

R.A. Fisher. UCI machine learning repository: Iris data set, 1936. URL `https://archive.ics.uci.edu/ml/datasets/iris`.

Development Team, Qiskit Machine Learning. Machine learning tutorials, 2021. URL `https://qiskit.org/documentation/machine-learning/tutorials/index.html#machine-learning-tutorials`.

Qiskit Development Team. qgans for loading random distributions, 2021a. URL `https://qiskit.org/documentation/machine-learning/tutorials/04_qgans_for_loading_random_distributions.html`.

Qiskit, IBM Quantum Experience. Introduction to quantum computing and quantum hardware, 2020b. URL `https://www.youtube.com/playlist?list=PLOFEBzvs-VvrXTMy5Y2IqmSaUjfnhvBHR`.

Qiskit Development Team. Feature maps, "Aug" 2021b. URL `https://qiskit.org/documentation/apidoc/qiskit.aqua.components.feature_maps.html`.

Nathan Wiebe, Ashish Kapoor, and Krysta Svore. Quantum algorithms for nearest-neighbor methods for supervised and unsupervised learning, 2014.

Kevin Krsulich. private communication: Qiskit Github, March. 2021.

SAT09 Conference. Sat competition 2009: Benchmark submission guidelines, "Jan" 2009. URL `http://www.satcompetition.org/2009/format-benchmarks2009.html`.

Qiskit Development Team. Introduction to quantum error correction using repetition codes, 2021c. URL `https://qiskit.org/textbook/ch-quantum-hardware/error-correction-repetition-code.html`.

D. Willsch, M. Willsch, H. De Raedt, and K. Michielsen. Support vector machines on the d-wave quantum annealer. *Computer Physics Communications*, 248:107006, 2020. ISSN 0010-4655. doi: https://doi.org/10.1016/j.cpc.2019.107006. URL `https://www.sciencedirect.com/science/article/pii/S001046551930342X`.

Baidu. These five ai developments will shape 2021 and beyond. *URl: https://www.technologyreview.com/2021/01/14/1016122/these-five-ai-developments-will-shape-2021-and-beyond/*, January
2021.

Mike Wheatley. Amazon proposes novel approach to quantum computing error correction. *URL: https://siliconangle.com/2021/04/12/amazon-proposes-novel-approach-quantum-computing-error-correction-based/*, April
2021.

Tin Tomašić. The efficiency of the quantum search | how effective is grover's algorithm (quantum search) opposed to classical computer searching algorithms in terms of time complexity?, 07 2020.

Austin G. Fowler and Lloyd C. L. Hollenberg. Scalability of shor's algorithm with a limited set of rotation gates. *Physical Review A*, 70(3), Sep 2004. ISSN 1094-1622. doi: 10.1103/physreva.70.032329. URL `http://dx.doi.org/10.1103/PhysRevA.70.032329`.

Adrian Cho. The biggest flipping challenge in quantum computing. *Science*, 2020a. doi: 10.1126/science.abd7332.

Adrian Cho. The biggest flipping challenge in quantum computing. *Science*, July 2020b. URL `https://www.sciencemag.org/news/2020/07/biggest-flipping-challenge-quantum-computing`.