Pattern Recognition with Quantum Support Vector Machine(QSVM) on Near Term Quantum Processors.

by

Sajjad Ahmed
15301095

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science and Engineering

Department of Computer Science and Engineering
Brac University
April 2019

Declaration

It is hereby declared that

1. The thesis submitted is my/our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

Sajjad Ahmed
15301095

Approval

The thesis/project titled "Pattern Recognition with Quantum Support Vector Machine(QSVM) on Near Term Quantum Processors" submitted by

1. Sajjad Ahmed (15301095)

Of Spring, 2019 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on April 25, 2019.

Examining Committee:

Supervisor:
(Member)

Dr. Mahbub Majumdar
Professor
Department of Computer Science and Engineering
Brac University

Program Coordinator:
(Member)

Dr. Jia Uddin
Associate Professor
Department of Computer Science and Engineering
Brac University

Head of Department:
(Chair)

Dr. Md. Abdul Mottalib
Professor and Chairperson
Department of Computer Science and Engineering
Brac University

Abstract

Machine Learning is boosting up the advancement in the field of Artificial Intelligence these days. However, almost every machine learning algorithm contains an optimization problem to solve. Inspired by quantum mechanics, quantum computing is quite a promising approach to solve high complexity optimization problems significantly faster and more efficient than classical computers. In this paper, we have worked on a very fundamental supervised learning problem. First, we discuss an approach to map the classical feature points on a quantum computer. Then we propose a Quantum Support Vector Machine(QSVM) model that runs on near term superconducting quantum processors. We show that using quantum optimization it is possible to train a discriminative SVM model that is capable of recognising patterns.

Keywords: Quantum Computing, Machine Learning, Quantum Machine Learning, Support Vector Machine

Acknowledgement

Firstly, I thank the almighty Allah for enabling me to complete this thesis work. Secondly, I thank my thesis supervisor Prof. Dr. Mahbub Majumdar from the deepest part of my heart. It was a great opportunity for me to work under his guideline. He just not only guided me but also motivated me throughout the time while working on this thesis. Everything I have learned from him through courses like Discrete Math and Machine learning helped me a lot while working on this research work. This could not have been possible without that knowledge. In addition to that, I would like to thank all of my family, friends and well-wishers for supporting me and being there whenever I needed. Finally, I would like to thank BRAC University for providing necessary facilities and working environment and giving me the opportunity to finish the thesis and complete my Bachelor's degree.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Quantum Computing has the potential to change our perception of computation that we have now. It is because of its weird nature of computing approach. It has been shown that quantum computers can solve many complex problems more efficiently than classical computers. It is possible to solve some hard to solve the computational problem with quantum computers. Shor's algorithm which is to solve prime factorization is a good example of that. The possible speedup is significant for the journey of computation, as we are heading towards the end of Moor's law. It is becoming harder to embed more transistor in small space. Thus it is becoming so hard to improve the single core speed and it became a challenge to find an alternative computational option.

In contrast, Machine Learning which is a kind of Artificial Intelligence is changing our world dramatically. The application of machine learning has been improving our life significantly. But training a machine learning model is a computationally expensive task due to iterative high algorithmic complexity. Currently, the popular hardware choice for training machine learning model includes GPU, TPU, ASIC etc. These solutions are comparatively faster than training with CPU. But the common fact that all of these shares is that these are all classical computing method. They all follow the same traditional computational theory. They just as differ architecture wise.

Thus, our approach is to merge these two concepts together. Here we show that the possibility of training a machine learning model using near-term quantum computers.

1.2 Problem statement

Machine learning algorithms are heavily dependent on a huge amount of data. The machine learning models learn from the data by finding a generalized pattern among the data points. Thus we need to fit a good number of data points for the better model. However, all the data presently we have are represented for classical computational. On the other hand, the quantum computer requires a different type of mapping for the data. Another challenge is to work with the image dataset, as it requires more feature points(pixel points) while training.

1.3 Objective

In this thesis our main objectives are:

- Map the classical image data on Quantum computer.
- Train a discriminative Quantum Support Vector Machine model to solve the classification problem.
- Train with more feature points on a quantum computer.
- Analyze the learnability of the discriminative model.

1.4 Methodology

We demonstrate an approach to implement classical feature point data in a quantum computer. Then we describe a quantum computing architecture to develop a machine learning model. We experimentally run the Quantum Support Vector Machine(QSVM) algorithm in both quantum and classical device. In this experiment, the MNIST handwritten data set was used. Finally, we run our experiment in both simulation and quantum device and compare the results.

1.5 Outline

This thesis report has been organized as following:

- **Chapter 1:** A general introduction of our work. It also includes a brief overview of the whole work.
- **Chapter 2:** This chapter contains the background analysis of the problem. The chapter is very **important** for understanding the work shown in the rest of this paper. Here we discussed the basic and important topics of quantum computing and machine learning.
- **Chapter 3:** Here we have analyzed the previous work done in this field.
- **Chapter 4:** Firstly we discuss the QSVM algorithm. Then we discuss the workflow and the implementation.
- **Chapter 5:** The analysis of the experiment has been discussed here. Also provides the necessary justification for the experiment.
- **Chapter 6:** Contains the conclusion of the report. It includes the future plan of the references used in this entire report.

1.6 Workflow

The workflow plan that we followed while working on this thesis is given below:

Research and planning

At this primary stage we divided the task into two parts:

1. **Literature Review**

   Quantum computing is an active research field. The literature review includes doing research about previous works done on this field. Online libraries, journals and institutional research papers are a great source of doing research about this field.

2. **Recent Research works**

   Quantum computing technology is changing over time. Coping up with current technologies, advancements is a very important part of this project.

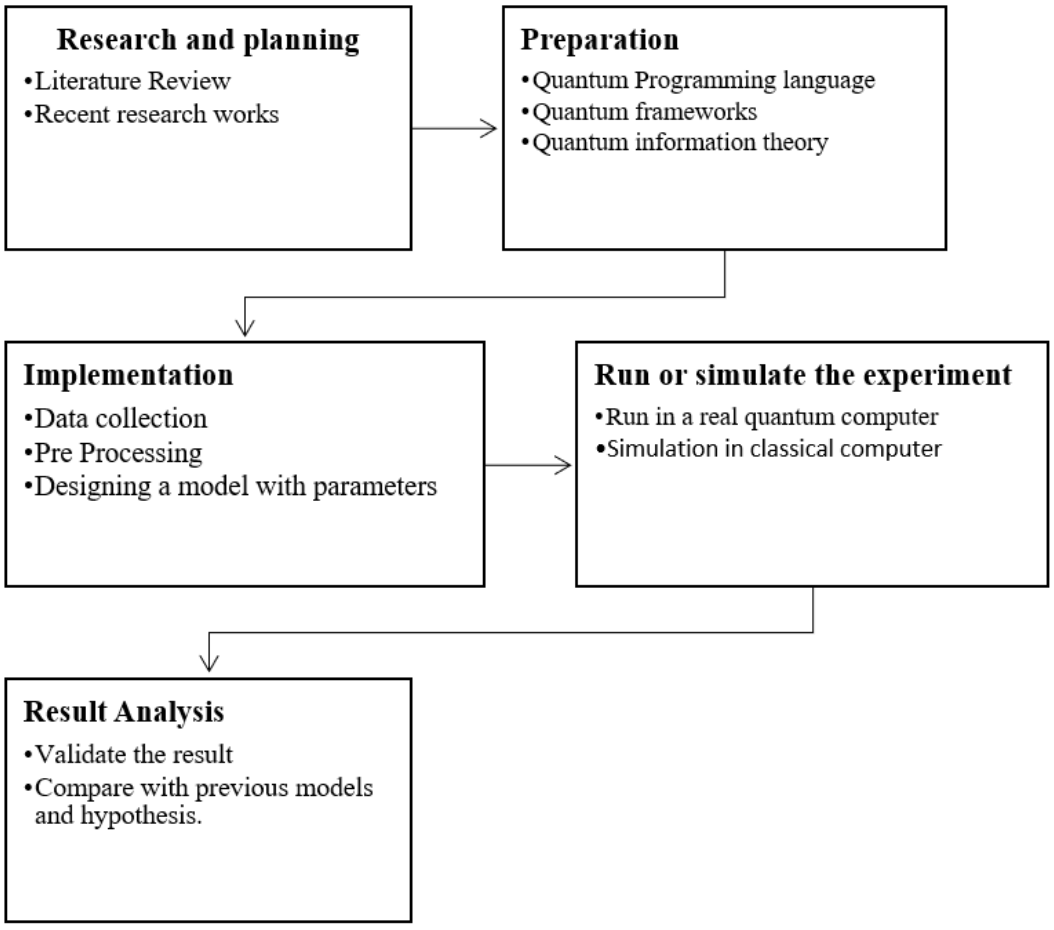

Figure 1.1: Work flow plan

Preparation

At this preparation stage, we mainly focused on skill development in this field. Three important fields are :

1. **Quantum programming language**

   Due to recent advancement in this field, a lot of high-level languages has been developed. With these languages, we can design quantum circuits with quantum logic. Topmost quantum language includes Q# (q sharp) and Quantum Computation Language (QCL). Apart from these, python is also supported in a few quantum frameworks

2. **Quantum frameworks**

   Quantum frameworks allow us to run on real open source quantum computers or simulate our models in our local computer. There are so many popular frameworks. But most popular among them are IBM Qiskit, Rigette pyQuil or Microsoft Quantum development kit. These frameworks are built on top of current popular high-level languages. For instance, Python.

3. **Quantum information theory**

   Quantum computing is an active research field. The contents and methodologies are very complex and critical. To work on this field, we need to spend some time to sharpen our knowledge. There are several online open courses from MITx.

Implementation

1. **Data collection**

   To train a machine learning model we need to collect data. Data can be collected from various open sources and repositories. For example UCI, Kaggle etc. In our case, we have to collect MNIST dataset[7].

2. **Data Pre-processing**

   Depending on the data-set we may need to pre-process the data before we fit into our model. This process includes scaling, labelling, removing anomaly and null points. Additionally, data should be split into two parts for training and testing purpose.

3. **Designing the model**

   After getting all the parameters from the data-set, we have to have designed a QSVM model. Here main challenges are designing the architecture in a quantum computing environment, finding a feature map and choosing the best parameters to get the best result.

Run or simulate the experiment

Due to rising hype of quantum computing we have these following options to run our quantum experiment.

1. **Run in a real quantum computer**

   There are three companies who have a quantum computer right now. They are IBM, D-Wave and Rigetti. The IBM quantum computer named IBM Q has 4-5 topological qubits. On the other hand, D-wave leap has 2000 qubits quantum computer. On the other hand, Rigetti has introduced the concept of Quantum Processing Unit(QPU) which is basically silicon-based annealing. Therefore, the theory behind these machines is different. That means the result may vary based on the machine. For our experiment, we will try to run our experiment on all of these devices.

2. **Simulation in classical machines**

   Based on topological qubit method we can simulate a quantum operation on a regular computer. The no of qubits depends on the ram size. For example, a machine with 8GB of RAM can simulate 29 qubits. A single qubit requires the exponential amount of RAM However, the result may vary slightly than a real quantum device due to the noise issue. A real quantum computer generates noise during the output where simulation may not produce any noise.

Result analysis

1. **Validate the result**

   After getting the result we have to validate its feasibility and accuracy. General machine learning analysis has been done on our model. Depending on the learnability and stability we have to finalize our result.

2. **Compare with hypothesis**

   After getting our result we have to compare with our hypothesis whether we can optimize and speed up our classical algorithms. If it optimizes than classical machines then our hypothesis will be proved.

1.7 Timeline

We have scheduled the timeline of our project for better productivity. We were managed to do it on time as well. The timeline is shown in Figure 1.2

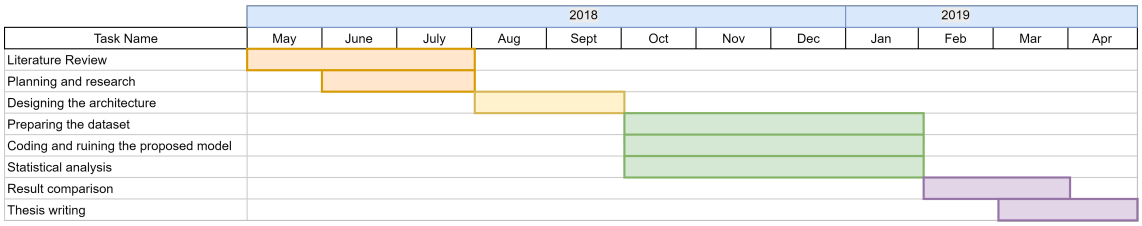

Figure 1.2: Thesis timeline

Chapter 2

Background analysis

2.1 Quantum computing

This section contains a basic introduction to the Quantum computing concepts and relevant subjects which is important to understand the work presented in this report.

2.1.1 Cbit and Qubit

The term Cbit refers to "Classical bit" used as the information unit in a classical computer or a two-state digital machine. A classical computer follows the binary format which is represented by strings of *One* and *Zero*. For example, the number 243 can be written as 11110011 in binary format and all sort of data can be encoded in this format according to the digital logic. Basically, these *One* and *Zero* are different individual states which construct a state machine. The state machine is the core concept of a classical computer's architecture. In general, at the circuit level, this can be achieved using a transistor. In a transistor, a higher voltage represents 1 state and lower voltage as 0 state. All the digital logic gates ex: AND and OR gates (Figure 2.1 2.2) can be designed in this way.

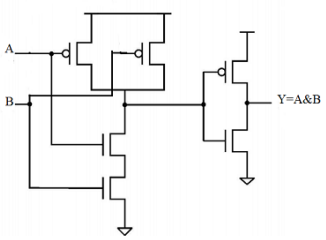

Figure 2.1: Circuit design of AND logical gate

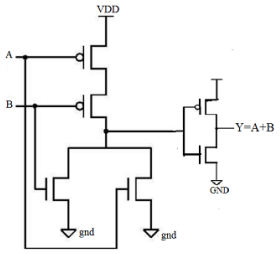

Figure 2.2: Circuit design of OR logical gate

In contrast, the basic unit of quantum information theory is called Qubit which stands for a Quantum bit. Unlike a classical bit, the qubit works a little bit weirdly. Qubit is defined by the probability of being in a state. Thus, a qubit has states like *One*, *Zero* and a third state called *Superposition*. All of these states can be represented as the unit vector in 2D complex vector space $\mathbb{C}^2$. The notation

used for a qubit is called Dirac notation. The *Zero* state can be written as $|0\rangle$ and *One* state can be written as $|1\rangle$. We consider them basis vectors in a 2D vector space denoted as $\mathbb{C}^2$. As 2D vector we represent $|1\rangle$ and $|0\rangle$ as

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

We can use the Bloch sphere in figure 2.3 to explain state of an Qubit. Here the south pole represents $|1\rangle$ state and the north pole represents $|0\rangle$. Apart form

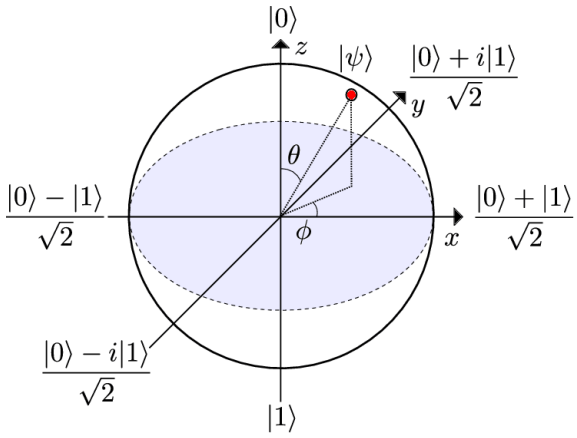

Figure 2.3: The Bloch sphere representing a qubit[8]

.

that a qubit can be in other states like other ray's of the sphere. Therefore, the any other state can be defined as

$$|\psi\rangle = \alpha_0.|0\rangle + \alpha_1.|1\rangle \tag{2.1}$$

Here $\alpha 0$ and α_1 are some complex number at point (θ, ϕ). We define $\alpha 0$ and α_1 as

$$\alpha_0 = \cos\frac{\theta}{2}$$

$$\alpha_1 = e^{i\phi}\sin\frac{\theta}{2}$$

We will come back to this equation in the next section. For now, this is the basic concept of representing a qubit in geometrical format.
To simplify this concept we can relate this to the example of coin flipping. In Cbit a coin has just two states represented by Heads and Tails. In terms of Qubit, along with the Heads and Tails we have the flipping state as the superposition state. Thus a question arises that if the coin the flipping state is not a certain state since it has both possibility to be at the both of the states. so how can we turn this uncertain state to a defined state.

2.1.2 Measuring a Qubit

So far we know a qubit can exist in a continuum state between $|0\rangle$ and $|1\rangle$ which is probabilistic. But we can turn it to a certain state by measuring its state which turns to either 0 or 1 state as a result. So let's see how this measurement works. First, let us assume that a qubit is in a certain state where it has 50% of the chance of being measured as 0 state and 50% chance of being measured in 1 state. It can also be denoted as $|+\rangle$ and can be written as

$$\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \tag{2.2}$$

But here it is important to understand how we think of a qubit in a physical system. We can realize a qubit as the spin of the electron where we can use it's spin to determine the state. We can also realize as ground and excited state for determining the state of a qubit. But whatever the case is we have to measure a qubit to determine it's state. From the last section, we can rewrite the equation 2.1 as

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle \tag{2.3}$$

Here, θ, φ and γ are real numbers. Now referring back to 2.3, there can be infinite amount of point we can get on the sphere. Thus paradoxically there can be infinity amount of information we can store in an qubit. But this statement gets eliminated if we consider the a qubit can be in only $|0\rangle$ or $|1\rangle$ state if measured. Intuitively, we can say that measurement can change a qubit state. For instance, if a qubit can be in $|+\rangle$ state but will change to either $|0\rangle$ or $|1\rangle$ when measured.

If a qubit is described as $\begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix}$ where α_0 and α_1 are complex numbers. Then we can find out each of the classical probability by using Born rule

$$p(x) = |\alpha_x|^2 \tag{2.4}$$

It refers the probability of occurring event. x Now according to basic probability rule we can write

$$|\alpha_0|^2 + |\alpha_1|^2 = 1 \tag{2.5}$$

Thus we can write the combination of basis vector to illustrate the superposition state. For multiple qubits, we can write

$$\alpha_{00}.|00\rangle + \alpha_{01}.|01\rangle + \alpha_{10}.|10\rangle + \alpha_{11}.|11\rangle = \begin{pmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{10} \\ \alpha_{11} \end{pmatrix} \tag{2.6}$$

2.1.3 Quantum gates

To represent Quantum gates we use matrices to show the changes on a qubit. In all the kinds of operations the normalized equation 2.5 must hold for all states. Thus it produces unitary vectors when we perform an action on a qubit. Due to unitary matrices, all the transformations are reversible. Thus all the quantum gates are

reversible. Which means the circuit should be able to run in reverse. Here if a given matrix is $\mathbf{U}$ then the conjugate transpose is denoted as $\mathbf{U}^\dagger$.

$$\mathrm{U}\mathrm{U}^\dagger = \mathbf{U}^\dagger\mathbf{U} = 1 \tag{2.7}$$

Hadamard gate

One of the most useful gate in quantum computing is called Hadamard gate. The Hadamard gate performs a rotation π about the X-axis and $\pi/2$ about the Y-axis in the Bloch sphere. This transformation takes X to Z and Z to X. The gate is among other things used to put the target qubit into a superposition state, having an equal chance of being measured as 0 or 1.

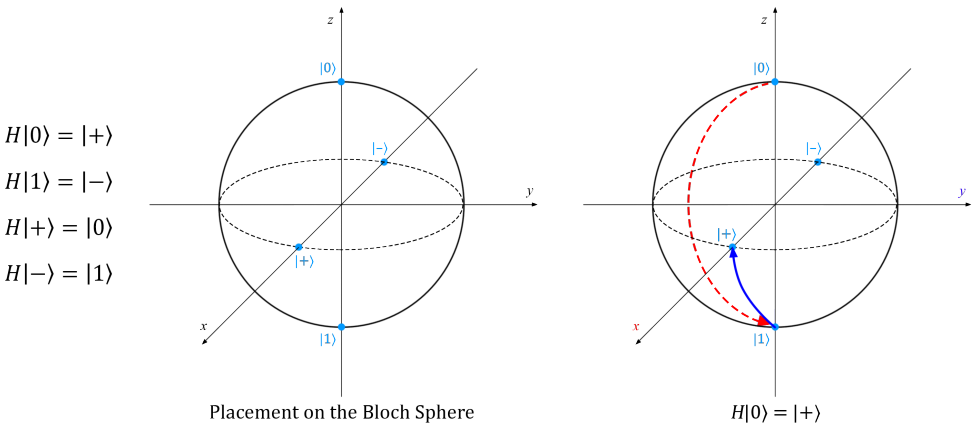

Figure 2.4: Hadamard gate representation in Bloch sphere [18]

$$\mathbf{H} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{2.8}$$

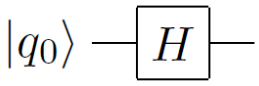

Figure 2.5: Hadamard gate matrix and block diagram

Controlled gate

Controlled gate is basically two qubits operation. The state of the first gate depends on the second gate. This can be used in other gates as well. We denote a controlled gate by adding a **c** is to the gate's name. For instance: **cNOT** for controlled not (cNOT gate). A **cNOT** gate is equal to a **cX**-gate. In the diagram below $|q0\rangle$ should be the control qubit and $|q1\rangle$ the target qubit. $|q1\rangle$ will change if the $|q0\rangle$ is

1. The **cNOT** matrix can be described by

$$\text{cNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.9}$$

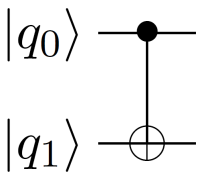

Figure 2.6: CNOT gate matrix and block diagram

Pauli gates

There are three types of Pauli gates, the X-, Y- and Z-gate. The gate rotates the qubit around the named axis in the Bloch sphere by π Figure 2.7. The X-gate is called bit-flip and the Z-gate phase-flip. The Y-gate is both a phase- and bit-flip and satisfies $Y = iXZ$ [6].

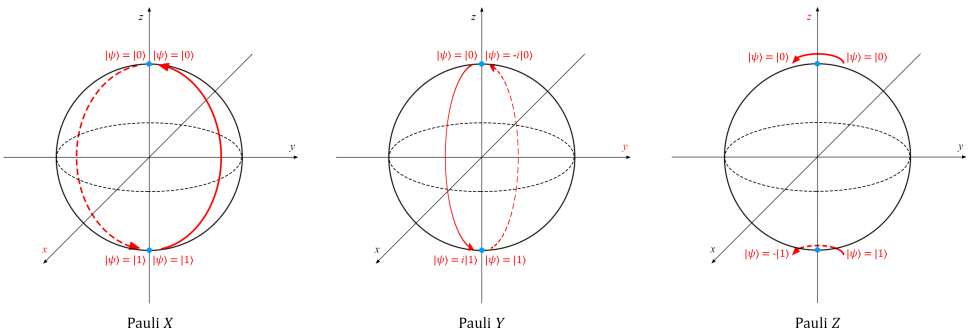

Figure 2.7: Pauli gate representation in Bloch sphere [18]

We represent Pauli gates as this following matrix and block diagram:

$$\mathbf{X} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad |q_0\rangle - \boxed{X} -$$

$$\mathbf{Y} = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \qquad |q_0\rangle - \boxed{Y} -$$

$$\mathbf{Z} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \qquad |q_0\rangle - \boxed{Z} -$$

Figure 2.8: Pauli gate block diagram [18]

T-gate

The T-gate is used for phase shifting the Pauli Z-gate. $T^4 = Z$, meaning that performing a T-gate four times will yield the same result as applying a Z-gate once. The T-gate corresponds to a rotation of $\frac{\pi}{4}$ around the Z-axis in the Bloch sphere. Shown in figure 2.9.

$$\mathbf{T} = \begin{pmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{4}} \end{pmatrix} \qquad |q_0\rangle - \boxed{T} -$$

Figure 2.9: T-Gate

Toffoli gate

A double controlled not gate is called a Toffoli gate. The Toffoli, ccNOT, and ccX gates are equivalent. In the diagram $|q2\rangle$ is the target qubit.

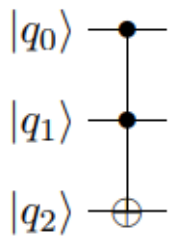

Figure 2.10: Toffoli Gate

Measurement gate

The measurement gate is actually not a quantum gate since it sets a value based on the observation. It is also a non-reversible operation. Because it sets the quantum states equivalent to the base vector that represents the measured state.

2.1.4 Two Qubit Operations

Representing Two Qubits

In quantum computing single qubit computation is a bit different than two or multi qubit operations. The core difference is two qubit states are 4 dimensional instead of two dimensional. This happens due to the two or multi qubit computation is formed by the tensor product on one qubit state. For example let's assume we have

$$00 \equiv \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad 01 \equiv \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$10 \equiv \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \quad 11 \equiv \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Therefore we can easily guess that the n qubit is represented by a unit vector of dimension 2^n using this construction.

$$\begin{bmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{10} \\ \alpha_{11} \end{bmatrix}$$

where

$$|\alpha_{00}|^2 + |\alpha_{01}|^2 + |\alpha_{10}|^2 + |\alpha_{11}|^2 = 1$$

Thus form above derivation we can generalize the two qubit operation as following:

Let's define our first qubit as $\begin{bmatrix} \alpha \\ \beta \end{bmatrix}$

and second qubit as $\begin{bmatrix} \gamma \\ \delta \end{bmatrix}$

Then the two qubit state is:

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix} \otimes \begin{bmatrix} \gamma \\ \delta \end{bmatrix} = \begin{bmatrix} \alpha \begin{bmatrix} \gamma \\ \delta \end{bmatrix} \\ \beta \begin{bmatrix} \gamma \\ \delta \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \alpha\gamma \\ \alpha\delta \\ \beta\gamma \\ \beta\delta \end{bmatrix} \tag{2.10}$$

here the operation $\otimes$ is called the tensor product (or Kronecker product) of vectors.

In terms of single-qubit, unitary transformation is valid. A unitary transformation on n qubits is a matrix U of size $2^n x 2^n$. For instance a common example of CNOT of two qubit gate is given below:

$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.11}$$

Now if we generalize this operation by assuming, the gates

$$\begin{bmatrix} ab \\ cd \end{bmatrix} \\ \begin{bmatrix} ef \\ gh \end{bmatrix} \tag{2.12}$$

to the first and second qubits, respectively, this is equivalent to applying the two-qubit unitary given by their tensor product:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \otimes \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae & af & be & bf \\ ag & ah & bg & bh \\ ce & cf & de & df \\ cg & ch & dg & dh \end{bmatrix} \tag{2.13}$$

2.1.5 Multi Qubit System

The multi qubit operation is quite similar to the two qubit operation. Here we also from tensor product of smaller states. For instance, for encoding a bit string 1011001 in computer we do the following way:

$$1011001 \equiv \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{2.14}$$

The quantum gates works also in the same way. For instance, if we want to apply X gate to the 1st qubit and then apply CNOT get in between 2nd and 3rd qubits, then we will end up with this following transformation:

$$(X \otimes \text{CNOT}_{12} \otimes \mathbf{1} \otimes \mathbf{1} \otimes \mathbf{1}) \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 0011001 \tag{2.15}$$

trhtsgrs490-=52In multi-qubit systems, due to the memory issue it often allocates and de-allocates qubits which serve as temporary memory for the quantum computer. These kind of qubit is called an ancilla. By default we assume the qubit state is initialized to e_0 upon allocation. We further assume that it is returned again to e_0 before de-allocation. This assumption is important because if an ancilla qubit becomes entangled with another qubit register when it becomes de-allocated then the process of de-allocation will damage the ancilla. For this reason, we always assume that such qubits are reverted to their initial state before being released.

2.1.6 Quantum Devices

- **IBM Q 14 Melbourne:** The IBM Q 14 Melbourne is a 14 qubit quantum computer developed by IBM. The architecture and the design is shown in fig. 2.11 [23]. We have used this device for our experiment.

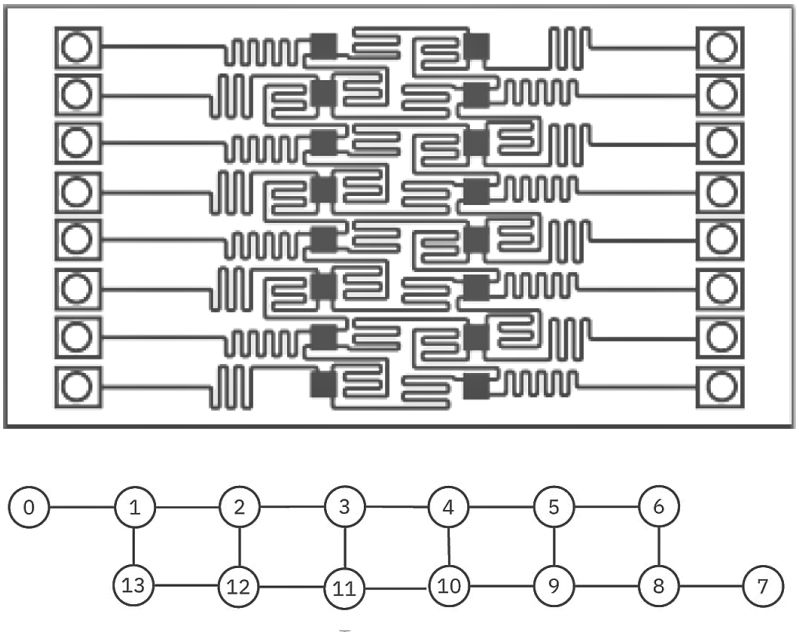

Figure 2.11: The 14 Qubit IBM Quantum computer(IBM Q 14 Melbourne) qubit mapping.

Table 2.1: Average measurements on IBM Q 14 Melbourne

Frequency (GHz)	5.10
T1 (μ s)	33.30
T2 (μ s)	37.90
Gate error (10-3)	3.01
Readout error (10-2)	3.49

- **IBM Q QASM Simulator:** As we already know that the benefit of quantum operation is the exponential speedup. Therefore it is possible to simulate the quantum operation in a classical device with a large amount of memory space. Although the result may vary due to the noise issue in the real quantum device. IBM has developed this IBM Q QASM simulator[23] that can simulate 32qubits operation. This simulator can take the algorithms and run them on IBM Q systems through the IBM Q Experience or the IBM Q Network without any code changes.

Qiskit

Qiskit[22] is an open-source quantum computing framework developed by IBM. It provides access to the quantum simulator and the quantum devices developed by IBM. This framework is based on the Python programming language.

The QISKit interface provides a built-in mapper for mapping a qubit in the code to a hardware qubit. Because of this, the qubits can be arbitrarily named in the QISKit code. The mapper works provided that the requested implementation can be made to fit the coupling map.

In our experiment, we have used another Qiskit library called Qiskit Aqua[21]. It provides a collection of all the cross-domain algorithms based on near term quantum devices.

2.2 Machine Learning

In this section, we have discussed the statistical analysis of Support Vector Machine and it's learnability. We start with the overview of VC Dimension and structural risk minimization. Then describe the basic ideas behind the Support Vector Machine (SVM) like how it learns using optimization.

2.2.1 Pattern recognition problem in Machine Learning

Let's assume that for l number of observations where each observation contains a pair of vector $\mathbf{x}_i \in R^n, \quad i = 1, \ldots, l$ and the given true label or the expected value are given by our source. In the recognition problem $\mathbf{x}_i$ might be a vector of pixel values e.g. n=784 for 28x28 pixel image and $\mathbf{y}_i$ can be the label e.g: 1.
Now it is expected that there exists some unknown probability distribution $P(x, y)$ form which data are drawn. In this case we consider IID(Independently drawn and Identically Distributed). Our trusted source will assign y_i according to a fixed distribution, conditional on x_i. But here we will be assuming a fixed y given by x.

Now our the job of our machine learning model is to learn mapping y_i given x_i, $\mathbf{x}_i \mapsto y_i$. Let's assume that our machine learning model came up with a function $\mathbf{X} \longmapsto f(\mathbf{x}, \alpha)$ where α is our predicted label. However is two possible outcomes, either $\mapsto y_i = \alpha$ or $\mapsto y_i \neq \alpha$. Which led us to expect some sort of error. Thus we define the test error of our machine as:

$$R(\alpha) = \int \frac{1}{2}|y - f(\mathbf{x}, \alpha)| dP(\mathbf{x}, y) \tag{2.16}$$

$R(\alpha)$ is called expected risk or the actual risk. It represents true mean error when density $p(x, y$ exists, $dP(\mathbf{x}, y)$ can be written $p(\mathbf{x}, y)d\mathbf{x}dy$.

Then we introduce another term called Empirical risk $R_{emp}(\alpha)$. Which is the mean error rate on the training set for a fixed and finite number of observations.

$$R_{emp}(\alpha) = \frac{1}{2l} \sum_{i=1}^{l} |y_i - f(\mathbf{x}_i, \alpha)| \tag{2.17}$$

Previously we defined the quantity $\frac{1}{2}|y_i - f(\mathbf{x}_i, \alpha)|$ as loss. For binary classification it can only take the values 0 and 1. Assuming some η (where $0 \leq \eta \leq 1$), for losses taking these values with probability $(1 - \eta)$ probability the following bound holds[3]:

$$R(\alpha) \leq R_{emp}(\alpha) + \sqrt{\left(\frac{h(\log(2l/h) + 1) - \log(\eta/4)}{l}\right)} \tag{2.18}$$

Here h is called the Vapnik Chervonenkis(VC) dimension which is a measure of the notion of capacity mentioned above. The right hand side of the equation 2.18 is called "risk bound". The lest part of the right hand side is called "VC Confidence".

2.2.2 VC Dimension

In general, VC dimension is a property of a set of functions $\{f(\alpha)\}$ which can be defined by various classes of function f. Let's consider a function that correspond to the binary pattern recognition problem $f(\mathbf{x}, \alpha) \in \{-1, 1\} \forall \mathbf{x}, \alpha$. If we have n points then we can label them 2^n possible way. Now for each labelling, a member of the set $\{f(\alpha)\}$ can be found which correctly assigns those labels then we say that set of the point is shattered by that set of function. The maximum number of the training points that can be shattered by $\{f(\alpha)\}$ is called the VC dimension for the set function $\{f(\alpha)\}$. If for a function the VC dimension is h, then there exist at least one set of h points that can be shattered.

2.2.3 Shattering Points with Oriented Hyperplanes in R^n

Let's assume that for a feature space $\mathbf{R}^2$, and the set of functions $\{f(\alpha)\}$ defined by a oriented straight line. Points of one side of the given line are labeled as class 1 and other points are as class -1. The orientation represents by the arrow in the figure 2.12 shows the points shattered by the line. Therefore it is possible to shatter three points. But not possible to find four points.

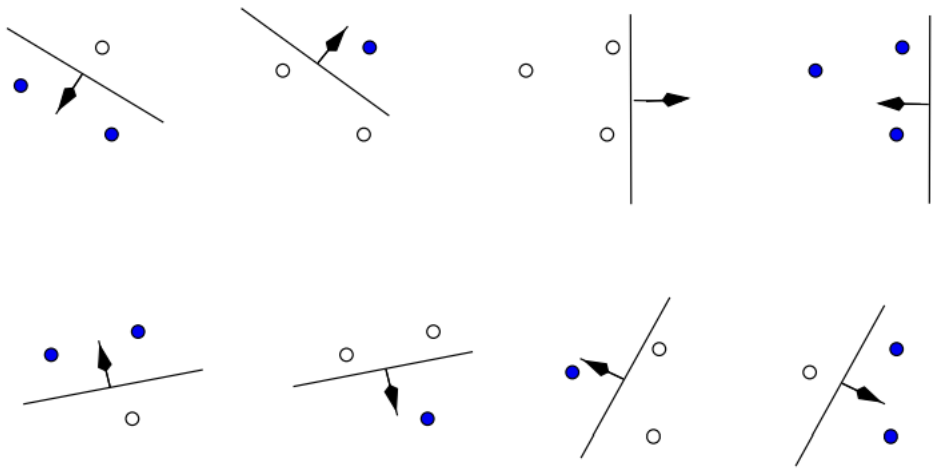

Figure 2.12: Three points($in\mathbf{R}^2$) shattered by line [3].

2.2.4 The VC Dimension and the Number of Parameters

Intuitively, we may think that having higher parameters would result in higher VC dimension and few parameters will result very low VC dimension. But this intuition is proven wrong by E. Levin and J.S. Denker (Vapnik, 1995). It's been stated that : A learning machine with just one parameter, but with infinite VC dimension (a family of classifiers is said to have infinite VC dimension if it can shatter l points, no matter how large l). The defination of step function $\theta(x), x \in \mathbf{R} : \{\theta(x) = \forall x > 0; \quad \theta(x) = -1 \forall x \leq 0\}$ [3]. Now lets consider one parameter family of functions defined by

$$f(x, \alpha) \equiv \theta(\sin(\alpha x)), \quad x, \alpha \in \mathbf{R} \tag{2.19}$$

Now we choose some number l number of points to be shattered:

$$x_i = 10^{-i}, \quad i = 1, \cdots, l \tag{2.20}$$

Then we specify the labels as:

$$y_1, y_2, \cdots, y_l, \quad y_i \in \{-1, 1\} \tag{2.21}$$

Then $f(a)$ gives this labeling if we choose α to be

$$\alpha = \pi \left(1 + \sum_{i=1}^{l} \frac{(1 - y_i)\, 10^i}{2} \right) \tag{2.22}$$

2.2.5 Minimizing The Bound by Minimizing h

For some of the machine learning algorithms which has empirical risk of zero, we want to choose that algorithm which has minimal VC dimension. It puts a better upper bound to the true error. In general, if the empirical risk is non zero, we would like to choose that machine learning algorithm which will minimize the right hand side of Eq 2.18.

Here we have to remember that we are only using Eq. 2.18 as a guide. Eq. 2.18 gives (with some chosen probability) an upper bound on the true error. It does not restrict a particular algorithm from the value of the empirical risk and whose set of function has higher VC dimension, from having better performance. In fact, it is possible that a system can perform better but can have infinite VC dimension. Here we can see that the graph shows that for $h/l > 0.37$ (and for $\eta = 0.05$ and $l = 10,\ 000$), the VC confidence exceeds unity, and so for higher values the bound is guaranteed not tight.

2.2.6 Linear Support Vector Machines

Here we begin with non linear separable case. Our task is to label the training data :

$$\{\mathbf{x}_i, y_i\}, \quad i = 1, \cdots, l, y_i \in \{-1, 1\}, \mathbf{x}_i \in \mathbf{R}^d$$

Now lets assume that we have some hyperplane that separates +ve and -ve points. The points where x lie in on the hyperplane satisfy $\mathbf{w} \cdot \mathbf{x} + b = 0$. Our goal is to find the margin that separates hyperplane d_+ and d_- where d_+ and d_- are the shortest distance from separating hyperplane. In this case the support vector machine algorithm will look for the hyperplany that has maximum margin. This can be derived in this following way:

$$\mathbf{x}_i \mathbf{w} + b \geq +1 \quad \text{for } y_i = +1 \tag{2.23}$$

$$\mathbf{x}_i \mathbf{w} + b \leq -1 \quad \text{for } y_i = -1 \tag{2.24}$$

Now if we combine Eq. 2.23 and 2.24 then we will get the following:

$$y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b \right) - 1 \geq 0 \quad \forall i \tag{2.25}$$

Now consider the points for which the equality in Eq. 2.23 holds (requiring that there exists such a point is equivalent to choosing a scale for w and b). These points lie on the hyperplane $H_1 : x_i \Delta w + b = 1$ with normal w and perpendicular

distance from the origin $|1-b|/\|\mathbf{w}\|$. Similarly, the points for which the equality in Eq. 2.24 holds lie on the hyperplane $H2 : xi\Delta w + b = -1$, with normal again w, and perpendicular distance from the origin $|-1-b|/\|\mathbf{w}\|$. Hence $d_+ = d_- = 1/\|w\|$ and the margin is simply $2/\|\mathbf{w}\|$. Note that H_1 and H_2 are parallel (they have the same normal) and that no training points fall between them. Thus we can find the pair of hyperplanes which gives the maximum margin by minimizing $\|\mathbf{w}\|^2$, subject to constraints 2.25 [3].

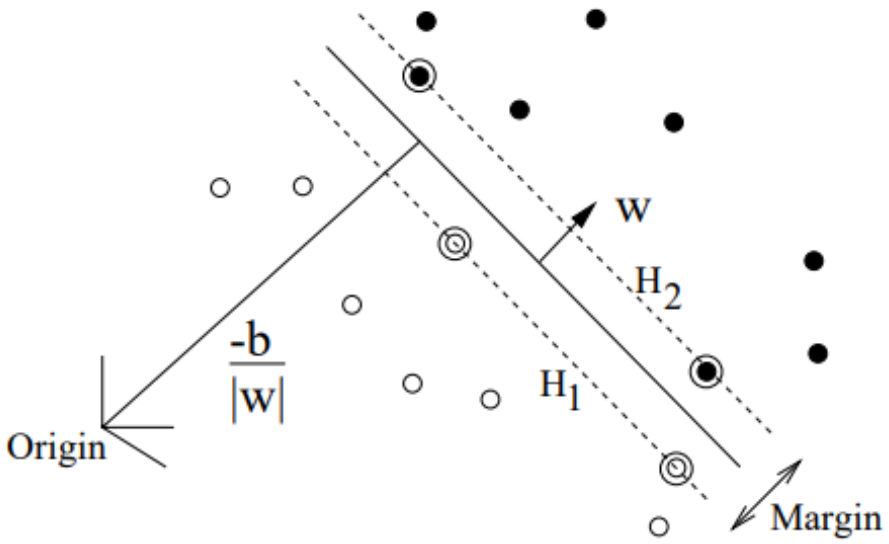

Figure 2.13: Linear separating hyperplanes for the separable case. [3]

Now as shown in Figure 2.13 a two dimensional case to solve. The training points lying on H_1 and H_2 hyperplane holds the equation no 2.25. If we are able to remove those points the solution can be found, then those points are called support vectors. Solving by Lagrangian formulationwe we get:

$$L_P \equiv \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{l} \alpha_i y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b\right) + \sum_{i=1}^{l} \alpha_i \tag{2.26}$$

Here in 2.26 our goal is to minimize L_p with respect to w and b. Requiring that the gradient of L_P with respect to w and b vanish give the conditions:

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i \tag{2.27}$$

$$\sum_i \alpha_i y_i = 0 \tag{2.28}$$

Since these are equality constraints in the dual formulation, we can substitute them into Eq. 2.26 to give

$$L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \tag{2.29}$$

In terms of separable linear case Support vector training, therefore, amounts to maximizing L_D with respect to the α_i, subject to constraints equation 2.28 and positivity of the α_i, with the solution given by equation 2.27. As found in this solution, those points for which $\alpha_i > 0$ are called "support vectors", and lie on one of the hyperplanes H_1, H_2. All other training points have $\alpha_i = 0$ and lie either on H_1 or H_2 or on that side of H_1 or H_2 such that the strict inequality in equation 2.25 holds. For these machines, the support vectors are the critical elements of the training set. They lie closest to the decision boundary; if all other training points were removed (or moved around, but so as not to cross H_1 or H_2), and training was repeated, the same separating hyperplane would be found.

Chapter 3

Literature Review

Over the years there has been a lot of demonstration regarding quantum computation. There are illustrations of how this emerging technology utilizes its effects to add a new dimension to this era of technological advancement. It opens new doors of immense possibilities for the researchers.

Jordan's algorithm[4] is one of the most significant algorithms in terms of optimization problems. The algorithm can predict the gradient estimation for a black box f and d a function with real variables for a given n precision. This algorithm requires only d number of black box query in quantum machines where it takes $d+1$ in classical machines. (Durr & Hoyer,1994) proposed a model[2] where they tried to find the minimum index in a table T of size N in time $O(\sqrt{n})$. (Gilyén, Arunachalam and Wiebe, 2017) they tried to develop a quantum algorithm that computes the gradient of a multi-variate and real-valued function $f : \mathbb{R}^d \to \mathbb{R}$ by evaluating only a logarithmic number of points in superposition[10]. Additionally, an improved version of Jordan's algorithm has been proposed which has the advancement of quantum gradient descent over classical gradient descent. Moreover, the research work has shown that for low-degree multivariate polynomials their algorithm can provide exponential speedups. They also proposed an improved version of Jordan's algorithm.

William Huggins, Piyush Patil, Bradley Mitchell, K. Birgitta Whaley, and E. Miles Stoudenmire have proposed that approaching to both discriminative and generative learning, quantum computing along with classical computing can benefit from the same theoretical and algorithmic developments[17]. In fact, it is possible to train the same model classically and then transfer it to the quantum setting for additional optimization. Having benefits for near term devices, tensor network circuits can provide qubit systematic schemes. In those schemes depending on the architectural design, the required number of physical qubits scales only logarithmically with, or independently of the input or output data sizes.

Zhikuan Zhao, Alejandro Pozas-Kerstjens, Patrick Rebentrost, and Peter Wittek in their research paper have stated that Bayesian strategies have great benefits in machine learning as they specifically provide estimates of the uncertainty related to a prediction[19]. The connection between deep feedforward neural networks and Gaussian processes allows the researchers to leverage a quantum algorithmic rule designed for Gaussian processes and develop a replacement algorithmic rule for Bayesian deep learning on quantum computers. They also aimed for demonstrating the execution of the algorithm on modern quantum computers. Their goal is to

analyze the hardiness of this algorithm with respect to realistic noise models.

Edward Farhi and Hartmut Neven have introduced a quantum neural network (QNN) which has the ability to represent labelled classical or quantum data. Again it can be trained by supervised learning[15]. Consisting of a sequence of parameter-dependent unitary transformations, the quantum circuit acts on an input quantum state. At the initial stage, they look at classifying classical data sets consisting of n-bit strings with binary labels. Here the input quantum state is such an n-bit computational basis state which corresponds to a sample string. They have shown a way to design a circuit made of two-qubit unitaries which will properly represent the label of any Boolean operation of n bits. They have introduced parameter dependent unitaries. These unitaries can be modified by supervised learning of labelled knowledge. Using classical simulation they have proven that the parameters which allow the QNN to learn to properly differentiate the two data sets can be found. For learning the label of a general quantum state their QNN can be used. According to their research, this QNN can be run on a near term gate model Quantum computer where the exploration of its power will be magnificent.

At the early 2000s, a good number of research works has been done on fundamentals and the possibility of applications of artificial intelligence in quantum machines. The concept of "quantum perceptron"[1] which is the fundamental of a single neuron was introduced in a paper by Lewenstein(1994) in his journal. Moreover, the author has shown that for low-degree multivariate polynomials their algorithm can provide exponential speedups compared to Jordan's algorithm in terms of the dimension d. He formulated a statistical theory of quantum perceptions, i.e. ideal quantum computing elements that process input states into output states through unitary transforms. (Ricks & Ventura, 2004) explained about "Training a quantum neural network"[5] in their research work. Their approach to training a Quantum Neural Network is to utilize the quantum speedup of search to find weights. They managed to solve a few popular machine learning problems like XOR, Iris and Gayes-Roth.

In 2016 Microsoft and Cambridge University researchers have implemented a few machine learning algorithms in a quantum environment and compared the complexity over a classical computer. As shown in Table 3.1, it's been found a significant speedup[9] in quantum methods. The key factor behind this speedup was the exponential number of states in quantum computers.

Method	**Speedup**
Bayesian Inference	$O(\sqrt{n})$
Online Perceptron	$O(\sqrt{n})$
Least squares ftting	$O(\log N(*))$
Classical BM	$O(\sqrt{n})$
Quantum BM	$O(\log N(*))$
Quantum PCA	$O(\log N(*))$
Quantum SVM	$O(\log N(*))$

Table 3.1: Quantum speedup in machine learning algorithms

In a recent research paper Daskin(2016) proposed e a simple neural network[14] that requires only $O(n \log(2k))$ number of qubits and O(nk) quantum gates. Here, n is the number of input parameters, and k is the number of weights applied

to these parameters in the proposed neural network. The numerical results indicate the network can be used in machine learning problems and it may provide exponential speedup over the same structured classical neural network. Apart from that, Bayesian deep learning algorithms are being implemented in quantum computers[19] and the Bayesian methods in machine learning, such as Gaussian processes, have great advantages compared to other techniques. In particular, they provided an estimation of the uncertainty associated with a prediction. However, extending the Bayesian approach to deep architectures has remained a major challenge. Recent research has shown that it is possible to connect deep feedforward neural networks with Gaussian processes, which allows training without backpropagation. The properties of the kernel matrix in the Gaussian process ensure the efficient execution of the core component of the protocol, quantum matrix inversion, providing at least polynomial speedup over the classical algorithm.

In addition, (Cong, Choi & Lukin, 2018) has proposed a Quantum Convolutional Neural Network(QCNN) [13] which uses only $O(\log(N))$ variational parameters for input sizes of N qubits and allowing for its efficient training and implementation on realistic, near-term quantum devices. Apart from that, the reinforcement learning (RL) algorithm has been proposed in a quantum computing environment. (López, Lamata, Retamal, C., Solano & E., 2018) have proposed a protocol to perform quantum reinforcement learning with quantum technologies[12]. They considered diverse possible scenarios for an agent, an environment, and a register that connects them, involving multi-qubits and multilevel systems, as well as open-system dynamics. According to their research, this will enable enhanced quantum control, as well as more efficient machine learning calculations. However, it is important to mention that all of these above-mentioned models are proposed at the theoretical stage.

Noise in quantum operation and output is a big challenge. According to (Verdon, Broughton & Jacob Biamonte, 2017) 's research on low depth circuits[11] has proposed a method which employs the quantum approximate optimization algorithm as a subroutine in order to approximate sample from Gibbs states of Ising Hamiltonians. They used this approximate Gibbs sampling to train neural networks for which they demonstrate training convergence for numerically simulated noisy circuits with depolarizing errors of rates of up to 4%.

After analyzing all these research works we found out that there are the enormous possibilities waiting for machine learning in quantum technology and there are a lot of scopes for improvement.

Chapter 4

Methodology and Implementation

This chapter we discuss how we implement our Quantum Support Vector Machine(QSVM) to train the discriminative model.

4.1 Dataset

In our experiment, we have used the MNIST dataset [7] with 60000 handwritten digits training set. From this dataset, we have used only two digits and a very small amount of data points to fit in the available near term quantum devices.

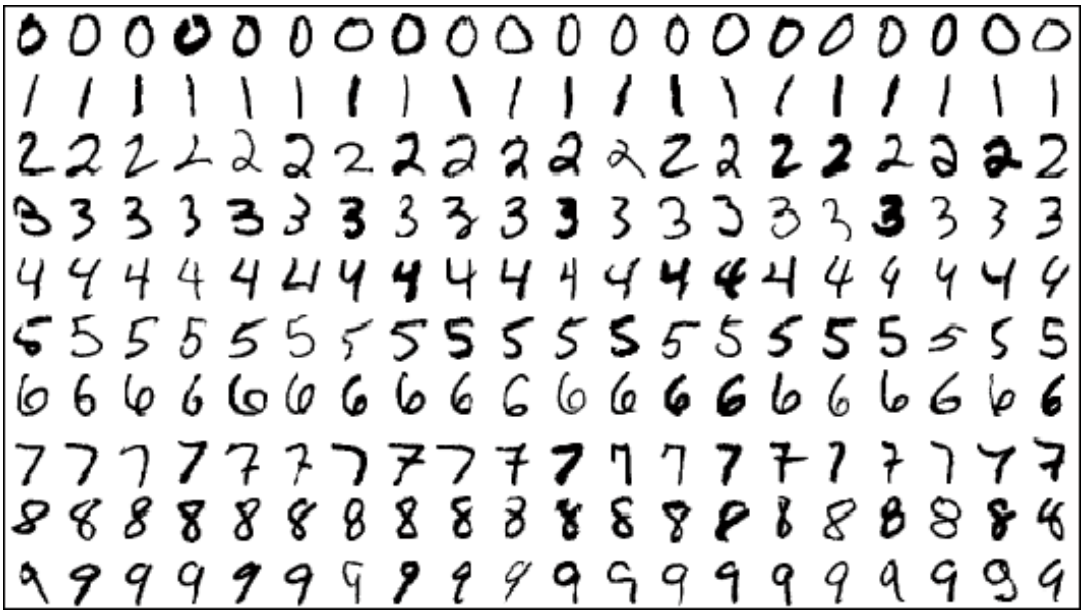

Figure 4.1: The MNIST dataset samples [7]

4.2 Pre-processing

After retrieving the data we followed these following pre-processing steps before training the model.

4.2.1 Data Splitting

At the initial stage, we had to split the data. Here we split the dataset in to two parts called training and testing part. The training set has the features and labels. This set is used for the model to learn. The other testing part has the feature points without any labels. By using this dataset we try to validate the accuracy by the model. The training part contains 80% of the data and the testing part contains 20% data.

4.2.2 Standard Scaling

At this stage, we remove the mean and scaling to unit variance for standardizing. Centring and scaling appear independently on every feature by computing the applicable statistics on the samples in the training set. The mean and popular deviation is then stored to be used on later data using the radically change method. Standardization of a dataset is a common requirement for many machine learning estimators. They would possibly behave badly if the individual points do not greater or much less seem to be like standard usually distributed data.

For example, many elements used in the objective function of a learning algorithm (such as the RBF kernel of Support Vector Machines or the L1 and L2 regularizers of linear models) expect that all features are established around 0 and have variance in the same order. If a character has a variance that is orders of magnitude large than others, it would possibly dominate the objective function and make the estimator unable to learn from different features effectively as expected.

4.2.3 Principal component analysis (PCA)

Principal component analysis compresses high dimensional data to lower dimensional data. Its basically dimensionality reduction using Singular Value Decomposition of the data to project it to a lower dimensional space.

It uses the LAPACK implementation of the full SVD or a randomized truncated SVD by the method of Halko et al. 2009, depending on the shape of the input data and the number of components to extract.

It can also use the scipy.sparse.linalg ARPACK implementation of the truncated SVD.

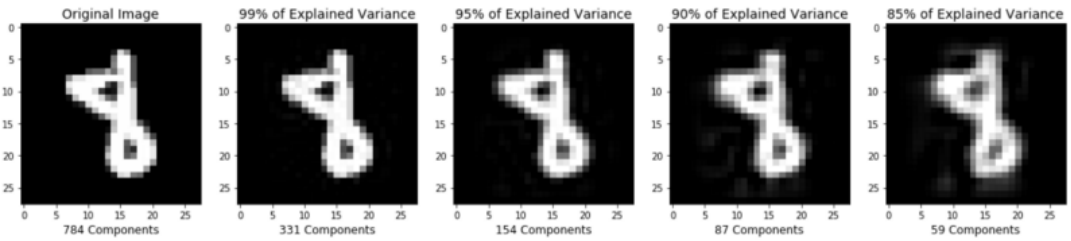

Figure 4.2: Principal component analysis

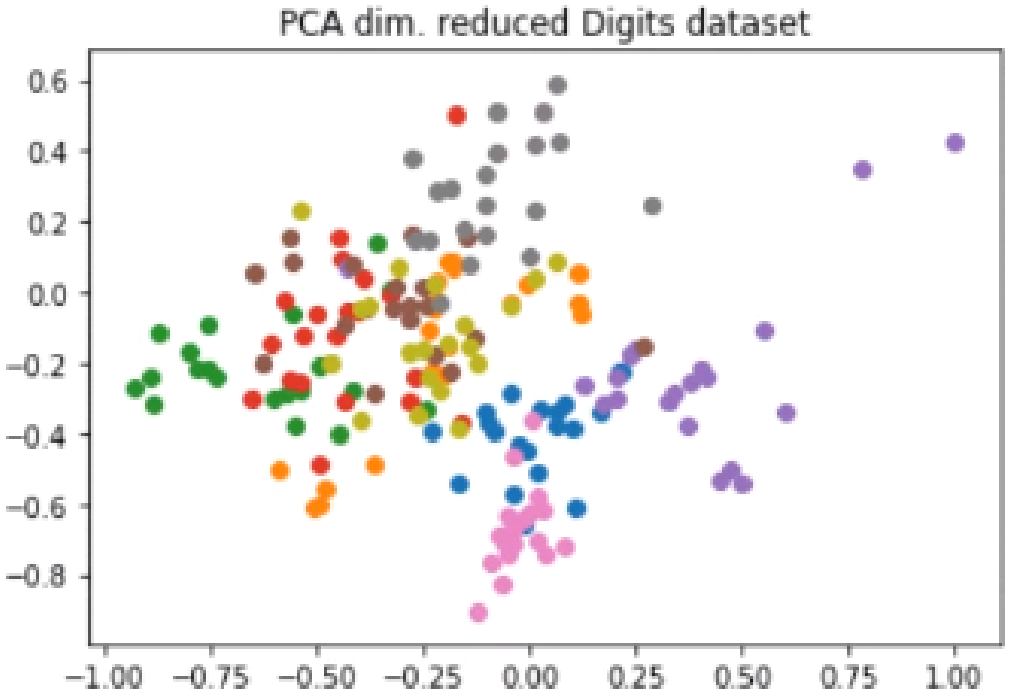

Figure 4.3: PCA dim reduced Digits dataset

Here the Fig. 4.3 shows the dimension reduced Digit dataset. However, we have chosen 2 digits which is 0 and 1 for our final experiment due to lack of available qubit.

4.2.4 Scaling the features range

As shown in the next section, the way the QSVM architecture derived, we needed to scale the feature points in the range of -1, +1.

By taking 4 features point, our dataset is shown in the following figure:

```
Out[7]: {'A': array([[-0.09738929, -0.37137445, -0.82812927,  0.15155979],
                [-0.04174598, -0.57091246, -0.68100516,  0.01060204],
                [ 0.08219813, -0.31884114, -0.45365309, -0.02775331],
                [-0.04656009, -0.28755953, -0.70515492,  0.06332981],
                [-0.02478349, -0.4303554 , -0.64652464,  0.08481711],
                [-0.08666301, -0.39014763, -0.66507369, -0.05136407],
                [ 0.01524102, -0.51000892, -0.56909607,  0.07261756],
                [ 0.02445676, -0.32876998, -0.80604283, -0.05824989],
                [-0.00801336, -0.44953474, -0.67048454,  0.07940728],
                [-0.01492937, -0.65497034, -0.86127548,  0.06053049],
                [ 0.21180812, -0.22293791, -0.58600321, -0.10243443],
                [ 0.06346287, -0.37774735, -0.77769966,  0.08453284],
                [-0.1676637 , -0.53652817, -0.70886145,  0.08402679],
                [ 0.10854304, -0.6100038 , -0.29323006, -0.12656763],
                [-0.22970361, -0.39261888, -0.75866715, -0.03407997],
                [ 0.16829041, -0.33639488, -0.61609042,  0.14236791],
                [-0.10117941, -0.34100168, -0.68618409, -0.0569758 ],
                [ 0.09915078, -0.38839052, -0.68416891,  0.1252857 ],
                [ 0.06307329, -0.3412999 , -0.45189316,  0.07724802],
                [ 0.21810669, -0.22024753, -0.33083136,  0.01832947]]),
         'B': array([[ 0.12114085, -0.0641356 , -0.61391784, -0.75398452],
                [-0.36492205, -0.48463103,  0.71754001, -0.93182577],
                [-0.17814452,  0.07229284,  0.71566474, -0.42880357],
                [-0.04460118, -0.03126306,  0.41418164, -0.2460701 ],
                [-0.4810433 , -0.55522865,  0.73266807, -0.42971128],
                [-0.21729459,  0.02926206,  0.71947137, -0.37049758],
                [-0.22137113, -0.01965272,  0.85419354, -0.32292569],
                [ 0.11481597,  0.05626393,  0.21625642, -0.66781164],
                [-0.19382931, -0.15663043,  0.74135786, -0.17221229],
                [-0.22345101, -0.04509432,  0.81669556, -0.30593717],
                [-0.19895544, -0.23253464,  0.56475851,  0.22692519],
                [-0.19248986,  0.08746704,  0.5312039 , -0.44052546],
                [-0.18365472,  0.08645084,  0.61890532, -0.50595715],
                [-0.50877159, -0.60794712,  0.5261649 , -0.63205734],
                [-0.22908024, -0.22339161,  0.55682617, -0.31135179],
                [ 0.11118344, -0.03179545,  0.07488866, -0.6930725 ],
                [-0.59125299, -0.50169453,  0.64777932, -0.30612719],
                [-0.49207374, -0.60370972,  0.71541242, -0.69050127],
                [-0.00961083,  0.02708118, -0.14014786, -0.73118366],
                [-0.23624935, -0.1070083 ,  0.61192026,  0.00767725]])}
```

Figure 4.4: Dataset after pre-processing with 4 feature points and 2 digits

Here the data is in dictionary format. The label "A" represents the digit "0" and the label "B" represents the digit "1". This data-set was prepared to fit in a 4 qubit machine. Due to the qubit limitation, we could only take a few data points.

4.3 Implementation

4.3.1 Quantum Support Vector Machine(QSVM)

The main goal of the support vector machine is to solve the classification problem. For a given training set T and test set S which are labelled by a map $\Omega \subset \mathbb{R}^d$. Our features are mapped in the range of

$$m : T \cup S \to \{+1, -1\} \tag{4.1}$$

The algorithm uses retrieves and processes the data in a classical way but training is done by the help of quantum state space. In terms of the first proposed model features are mapped in non linearly to a quantum state $\Phi : \vec{x} \in \Omega \to |\Phi(\vec{x})\rangle\langle\Phi(\vec{x})|$[16]. As shown in figure 4.6 we represent our classical feature data in the Bloch sphere[16].

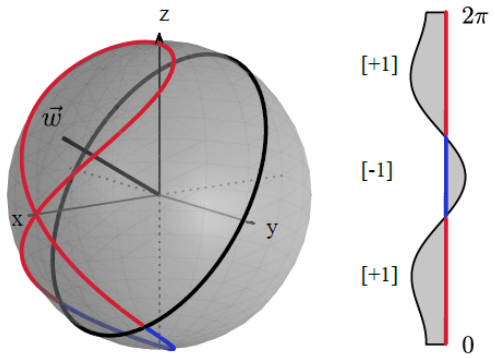

Figure 4.5: The feature map representation on Bloch sphere for a single qubit

This model is proposed for binary classification problem as the *Blue* and *Red* labels can be represented using this model in an interval of $\Omega = (0, 2\pi]$.
The second model which we used in our experiment is basically followed the unitary operation. Here each of the feature points is handled by a single qubit. The phase gate of angle $x \in \Omega$ is applicable ofr every qubit $U_{\Phi(x)} = Z_x$. We can distinguish the mapped-data by the hyperplane given by the $\vec{w}$. The positive values will be set to $[+1]Red$ and the negative values to $[-1]Blue$. In the circuit, the $U_{\Phi(\vec{x})}$ is formed by the products of single and two-qubit unitaries that are diagonal the computational basis[16].

One of the major challenge is to map the features. For mapping features for n qubits generated by the unitary

$$\mathbf{U}_\Phi(\vec{x}) = U_{\Phi(\vec{x})} H^{\otimes n} U_{\Phi(\vec{x})} H^{\otimes n}$$

where

$$U_{\Phi(\overline{x})} = \exp\left(i \sum_{S \subseteq [n]} \phi_S(\vec{x}) \prod_{i \in S} Z_i \right) \tag{4.2}$$

with this demonstration we plan to map and implement our QSVM model . The

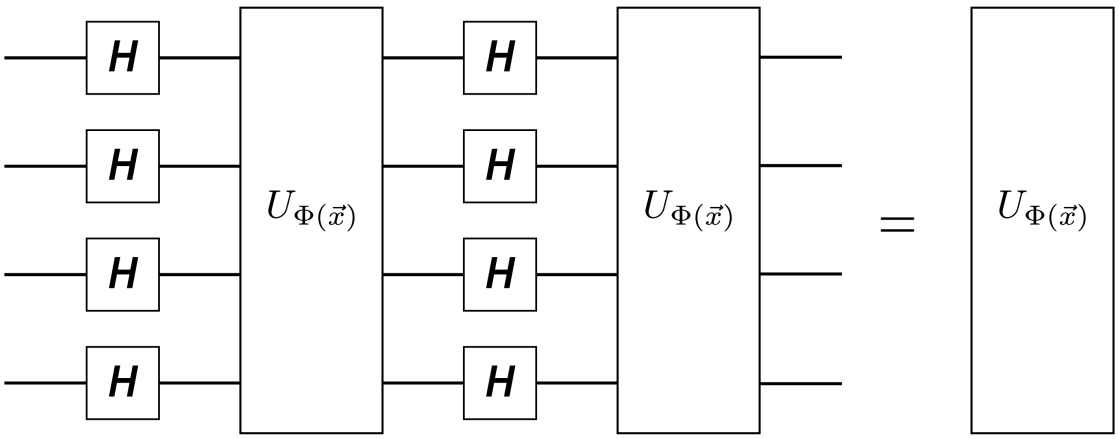

Figure 4.6: Features mapping with qubits.

final pseudo code for the QSVM model is given as following :

Algorithm 1: Pseudo-code for QSVM training phase

Input: training samples with labels $T = \{\vec{x}, y\}$where $\vec{x} \in \Omega \subset \mathbb{R}^n$ and $y \in C\}$

Parameters: $\vec{\theta}$ initial parameter
while $R_{\text{emp}}(\overrightarrow{\theta})$ *has not converged* **do**
  for $i = 1$ *to* $|T|$ **do**
    set $r_y = 0$ for every $y \in C$
    `/* here` r_y `is a counter */`

    for $shot = 1$ *to* R **do**
      prepare initial feature map state $|\Phi(\vec{x}_i)\rangle \langle\Phi(\vec{x}_i)|$ by using $\mathcal{U}_{\Phi(\vec{x}_i)}$
      apply discriminator circuit $W(\overrightarrow{\theta})$ to the initial feature-map state.
      get outcome measurement $\{M_y\}_{y \in C}$ by applying $|C|-$
      get measurement outcome label y by setting $\mathrm{r}_y \rightarrow \mathrm{r}_y + 1$
    end
    calculate empirical distribution $\hat{p}_y(\vec{x}_i) = \mathrm{r}_y R^{-1}$
    check the accuracy and error rate by evaluating
     $\Pr(\tilde{m}(\vec{x}_i) \neq y_i | m(\vec{x}) = y_i)$ with $\hat{p}_y(\vec{x}_i)$ and y_i
    $R_{\text{emp}}(\overrightarrow{\theta}) = R_{\text{emp}}(\overrightarrow{\theta}) + \Pr(\tilde{m}(\vec{x}_i) \neq y_i | m(\vec{x}) = y_i)$ `/* update cost function */`

  end
  Define new $\overrightarrow{\theta}$ with the data from $R_{\text{emp}}(\overrightarrow{\theta})$
end
return the final parameter $\overrightarrow{\theta}^*$ and value of the cost function $R_{\text{emp}}(\theta^*)$

4.3.2 Backend

The bookend refers whether we want to simulate the experiment in classical device or run in real quantum device. In our experiment we have implemented both of the options. In terms of simulation we used a regular classical computer. Then we

also used IBM Q QASM Simulator. Then we ran our experiment in a real quantum computer which is IBM Q 14 Melbourne.

4.3.3 Second Order Expansion

The Second Order Expansion feature map transform data $\vec{x} \in \mathbb{R}^n$ according to the following equation, and then duplicate the same circuit with depth **d** times, where **d** is the depth of the circuit [20]:

$$U_{\Phi(\overline{x})} = \exp\left(i \sum_{S \subseteq [n]} \phi_S(\vec{x}) \prod_{i \in S} Z_i\right) \tag{4.3}$$

where

$$S \in \{0, 1, \ldots, n{-}1, (0,1), (0,2), \ldots, (n{-}2, n{-}1)\}, \phi_i(\vec{x}) = x_i, \phi_{(i,j)}(\vec{x}) = (\pi - x_i){*}(\pi - x_j) \tag{4.4}$$

4.3.4 Feature Maps

For pattern recognition and image processing problem, a feature map starts from an initial set of measured data and incorporates features intended to be informative and non-redundant, facilitating the subsequent learning and generalization steps, and in some cases leading to better human interpretations.

In general what we do is basically dimension reduction. It involves reducing the number of resources needed to explain an oversized set of information. once applying the analysis of complicated data, one in every of the most important issues stems from the number of variables concerned. Analysis with an oversized range of variables usually needs a large quantity of memory and computation power, and will even cause a classification algorithm to over-fit to training samples and generalize poorly to new samples. once the input file to AN algorithm is just too large to be processed and is suspected to be redundant (for example, the identical measure is provided in each pound and kilograms), then it may be remodelled into a reduced set of features, named a feature vector. the method of deciding a set of the initial features is named feature choice. the chosen features are expected to contain the relevant info from the input file, so the specified task may be performed by using the reduced illustration rather than the entire initial data[20].

Entanglement

We define the entangler map with a dictionary of lists of non-negative int values :

$$entanglermap = 0 : [1|...|q-1], 1 : [0|2|...|q-1], ..., q-1 : [0|1|...|q-2]$$

The entanglement parameter defined above can be overridden by an entangler map explicitly specified as the value of the entangler map parameter if an entanglement map different from full or linear is desired. As explained more generally above, the form of the map is a dictionary; each entry in the dictionary has a source qubit index as the key, with the corresponding value being a list of target qubit indexes

to which the source qubit should be entangled [20]. Indexes are int values from 0 to $q-1$, where Q is the total number of qubits, as in the following example:

$$entanglermap = \{0 : [1, 2], 1 : [3]\}$$

Final Implementation

Before running the experiment these are the final parameter as shown below:

```
{'name': 'QSVM.Kernel',
 'description': 'QSVM_Kernel Algorithm',
 'input_schema': {'$schema': 'http://json-schema.org/schema#',
  'id': 'QSVM_Kernel_schema',
  'type': 'object',
  'properties': {},
  'additionalProperties': False},
 'depends': ['multiclass_extension', 'feature_map'],
 'problems': ['svm_classification'],
 'defaults': {'feature_map': {'name': 'SecondOrderExpansion', 'depth': 2}}}
```

Figure 4.7: Parameters and Configuration of QSVM

Here the **backend** varied while running on different machine. Now we implement these parameters and the result of our experiment has been shown in the next chapter.

Chapter 5

Result Analysis

5.1 Result analysis of Simulation on Classical Computer

In our experiment, we first ran a simulation on our classical computer. As discussed in the previous section, 4 feature points for our experiment. Here we discuss the basic machine learning result analysis.

Accuracy:

When we ran our algorithm with 4 feature point the accuracy we got was **95%**. The following work shows the general justification for our work.

- **Confusion Matrix:** In terms of Classification problem in machine learning, the performance of a particular learning model can be visualized by a table called confusion matrix table. In this table each row represents the instances in a predicted class by the model and each column represents the instances in an real class. With this relationship table we can calculate the performance. In our experiment we found the following confusion matrix:

```
Confusion Matrix
 [[10  0]
 [ 1  9]]
True negatives : 10
False negatives : 1
True positives : 9
False Positives : 0
```

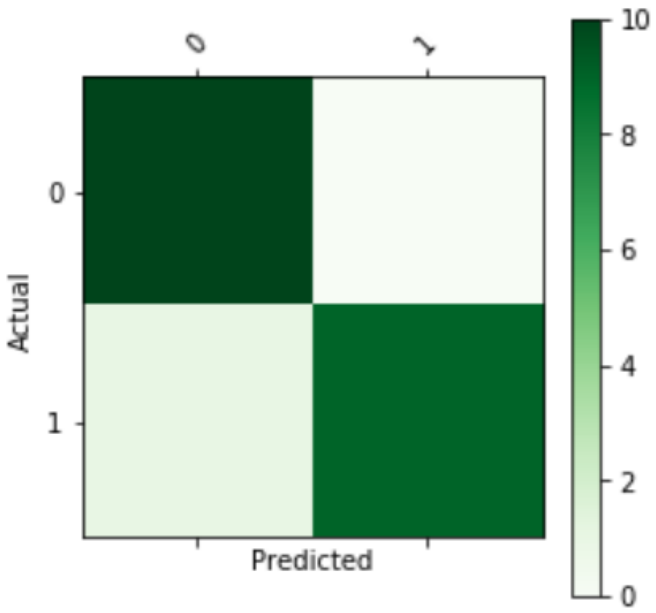

Figure 5.1: Confusion matrix of classical 4 feature point experiment

- **Sensitivity:** Sensitivity refers if we get a positive outcome then how often our prediction is correct. Higher the number the better our model is. It is also known as "Recall". We find a confusion matrix with the following equation:

$$\text{TPR} = \frac{\text{TP}}{P} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

  In our experiment, we found the sensitivity score of 0.9. Which is very high enough and a good score.

- **Specificity:** Specificity refers to when we get our prediction negative then how often our prediction is correct. We can find specificity with the following equation

$$\text{TNR} = \frac{\text{TN}}{N} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

  In our experiment, we got a specificity score of 1.0. Which means if we get a negative result then all the time our prediction is correct.

- **False Positive Rate:** False positive rate shows if we get a negative prediction then what is the probability of being our prediction is wrong. The less the value is the better our model. We can find False positive rate using this following equation:

$$\text{FPR} = \frac{\text{FP}}{N} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

  In our experiment, the false positive rate was 0.0. That means if we get a negative prediction there is 0% chance of getting the wrong output.

- **Precision:** In general, it refers to how "precise" the classifier is when predicting positive instances. In other words, how often our positive output is

correct. The higher the number the better the model is.

$$\text{PPV} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

We got a precision score of 1.0 in our experiment.

- **Kernel matrix:** The kernel matrix shows how the support vector machine algorithm can draw the margin to differentiate the classes. In this experiment, we can see our kernel matrix can label the classes properly most of the time.

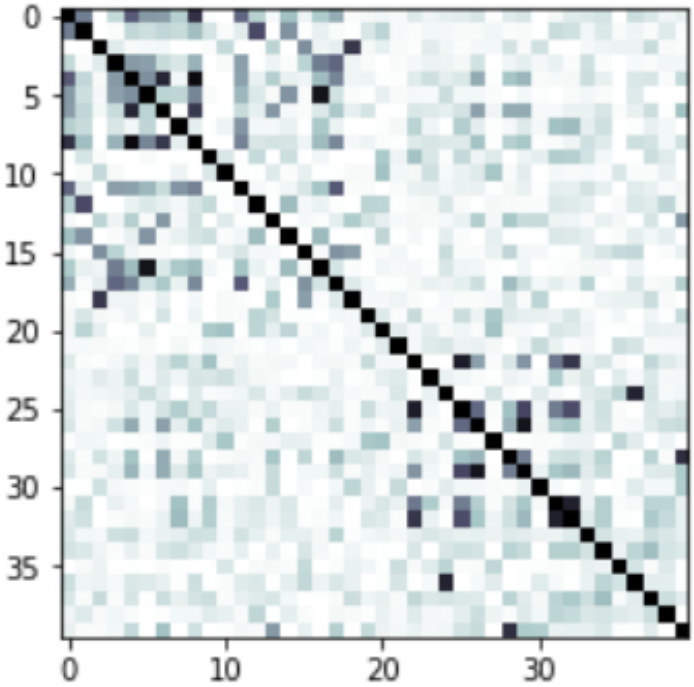

Figure 5.2: QSVM Kernel matrix on Classical computer

5.2 Result analysis of IBM quantum computer

At the next stage, we implemented and ran our algorithm on a real quantum device IBM Q 14 Melbourne. This is a 14 qubit quantum computer developed by IBM. The analysis of the result is given below:

Accuracy:

The accuracy we got was around **80%**. Here is the necessary analysis of our experiment:

- **Confusion Matrix:** In terms of Classification problem in machine learning, the performance of a particular learning model can be visualized by a table called confusion matrix table. In this table each row represents the instances in a predicted class by the model and each column represents the instances in an real class. With this relationship table we can calculate the performance. In our experiment we found the following confusion matrix:

```
Confusion Matrix
 [[8 2]
 [2 8]]
True negatives : 8
False negatives : 2
True positives : 8
False Positives : 2
```

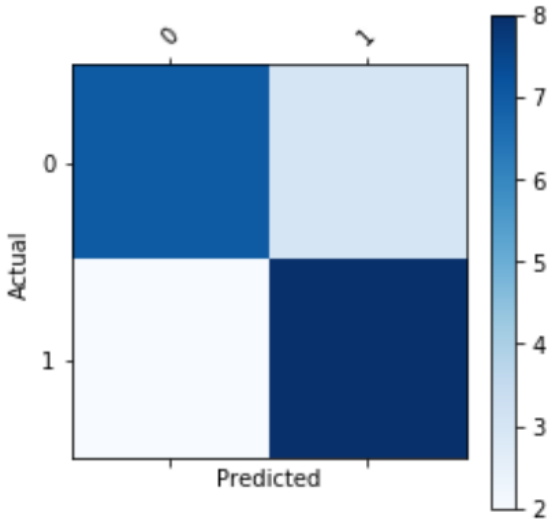

Figure 5.3: Confusion matrix of classical 4 feature point experiment in Quantum computer

- **Sensitivity:** Sensitivity refers if we get a positive outcome then how often our prediction is correct. Higher the number the better our model is. It is also known as "Recall".
  While running on a real quantum device we've got a sensitivity score of **0.8**.

- **Specificity:** Specificity defines when we get our prediction negative then how often our prediction is correct. In our experiment on the quantum machine we got specificity score of **.80** . Which means if we get a negative result then all the time our prediction is correct.

- **False Positive Rate:** False positive rate shows if we get a negative prediction then what is the probability of being our prediction is wrong. The less the value is the better our model. After running on a Quantum computer our FPR was **0.20**. Which means if we get a negative prediction there is 20% chance of getting the wrong output.

- **Precision:** In general, it shows how "precise" the classifier is when predicting positive instances. In other words, how often our positive output is correct. The higher the number the better the model is. We got a precision score of **0.80** in our experiment while running in a real quantum computer.

- **Kernel matrix:** The kernel matrix shows how the support vector machine algorithm can draw the margin to differentiate the classes. In this experiment, we can see our kernel matrix can label the classes properly most of the time. The kernel matrix has been shown in the following Figure 5.4

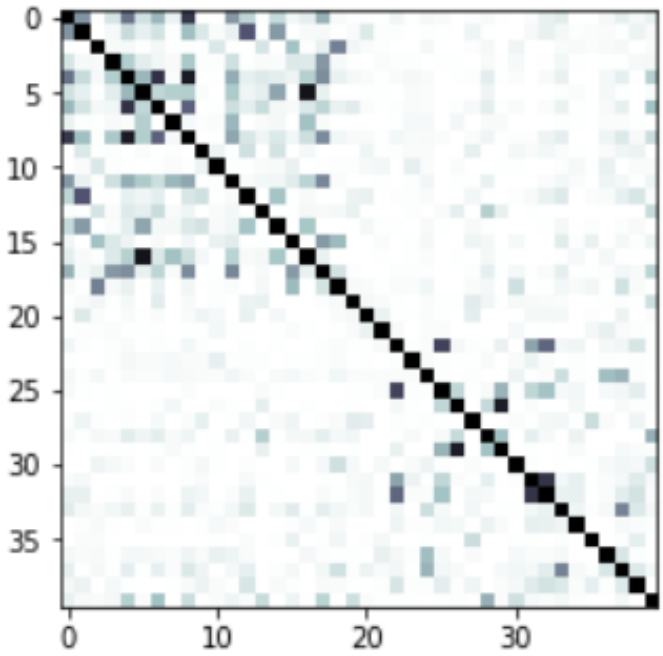

Figure 5.4: Kernel matrix on quantum computer

5.3 Result Comparison

As we can see from the analysis given above that the QSVM model was able to map the data in real quantum device. Moreover our model was able to learn from the data set and could classify new data as well. The accuracy comparison is given in the following table:

Device type	**Accuracy**
Simulation on classical computer	94.99%
IBM Q 14 Melbourne	80.04%

Table 5.1: QSVM accuracy on quantum and classical device

It turns out the quantum computer has less accuracy than the classical device. Which is completely valid and expected though. It happens due to the noise issue in a quantum computer. Removing noise in a quantum computer is one of the biggest challenges so far. Apart from the noise issue, we have implemented our QSVM model and recognise the patterns in both real and classical machine.

5.4 Limitations

We have found a few limitations of this experiment as well. First of all, as shown in the result the accuracy on the quantum computer is around 15% less than a classical computer. Here it is good to mention that the classical simulation is also implemented by mapping in quantum way. A general normal SVM would have performed better than this and by using kernel tricks it possible to boost up accuracy as well. Second of all, In this model, we have incorporated only four feature points for the four-qubit system. We have reduced the dimension of data by principal component analysis. In real life problems, there are more data points and we may need more qubit to map those. In addition to that, the model runs at the circuit level in the quantum computer. Therefore, it is not possible to modify on the fly or check each shot.

Chapter 6

Conclusion

6.1 Conclusion

Throughout our thesis work we tried to show the possibility of learnability in Quantum Computers. The big challenge was to map our classical data on quantum computer. We have experimentally implemented the QSVM on near term device that have access right now. Where we found the accuracy gap between the simulation and real quantum device and it happened due to the computational error. Which might be minimized in the future. However, the bridge that we tried to build between theoretical hypothesis and application layer is one of the core achievement of this work.

6.2 Future Work

Quantum computing is a growing and active research field. The technology is improving gradually as well as our theoretical research. Thus, there are a lot of scopes for future work in this field.

Noise in Quantum Computer is one of the biggest challenges right now. It also relative to the implementation of our quantum understanding which is based on physics. There are many elements that can cause noise. One of the main cause is the noise is caused by the very subtle measurements at the hardware level. The quantum computers needed to be cooled down at the coldest temperature in this universe. Slight vibration can cause a huge amount of noise in result. This is a challenge for the hardware designers to reduce the noise. If its possible to reduce the noise, the quantum algorithms can be significantly feaster and more accurate.

So far we have a decent number of qubits in the quantum computers. But to exceed the classical computer, there is an importance of increasing the number of qubits. If the number of qubits increases in the future, it will be more challenging to implement these algorithms. Therefore, there is room for improving these algorithms for large scale quantum devices.

Bibliography

[1] M. Lewenstein, "Quantum perceptrons", *Journal of Modern Optics*, vol. 41, no. 12, pp. 2491–2501, 1994. DOI: 10.1080/09500349414552331. eprint: https://doi.org/10.1080/09500349414552331. [Online]. Available: https://doi.org/10.1080/09500349414552331.

[2] C. Durr and P. Hoyer, "A quantum algorithm for finding the minimum", Jul. 1996. arXiv: quant-ph/9607014v2 [quant-ph]. [Online]. Available: http://arxiv.org/abs/quant-ph/9607014v2.

[3] C. J. Burges, "A tutorial on support vector machines for pattern recognition", *Data mining and knowledge discovery*, vol. 2, no. 2, pp. 121–167, 1998.

[4] S. P. Jordan, "Fast quantum algorithm for numerical gradient estimation", May 2004, Phys. Rev. Lett. 95, 050501 (2005). DOI: 10.1103/PhysRevLett.95.050501. arXiv: quant-ph/0405146v2 [quant-ph]. [Online]. Available: http://arxiv.org/abs/quant-ph/0405146v2.

[5] B. Ricks and D. Ventura, "Training a quantum neural network", in *Advances in Neural Information Processing Systems 16*, S. Thrun, L. K. Saul, and B. Schölkopf, Eds., MIT Press, 2004, pp. 1019–1026. [Online]. Available: http://papers.nips.cc/paper/2363-training-a-quantum-neural-network.pdf.

[6] N. Mermin, *Quantum Computer Science: An Introduction*. Cambridge University Press, 2007, ISBN: 9781139466806. [Online]. Available: https://books.google.com.bd/books?id=q2S9APxFdUQC.

[7] Y. LeCun and C. Cortes, "MNIST handwritten digit database", 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/.

[8] A. Frisk Kockum, "Quantum optics with artificial atoms", PhD thesis, Dec. 2014, ISBN: 9789175971131.

[9] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, "Quantum machine learning", Nov. 2016, Nature 549, 195-202 (2017). DOI: 10.1038/nature23474. arXiv: 1611.09347v2 [quant-ph]. [Online]. Available: http://arxiv.org/abs/1611.09347v2.

[10] A. Gilyén, S. Arunachalam, and N. Wiebe, "Optimizing quantum optimization algorithms via faster quantum gradient computation", Nov. 2017. arXiv: 1711.00465v3 [quant-ph]. [Online]. Available: http://arxiv.org/abs/1711.00465v3.

[11] G. Verdon, M. Broughton, and J. Biamonte, "A quantum algorithm to train neural networks using low-depth circuits", Dec. 2017. arXiv: 1712.05304v1 [quant-ph]. [Online]. Available: http://arxiv.org/abs/1712.05304v1.

[12] F. A. Cárdenas-López, L. Lamata, J. C. Retamal, and E. Solano, "Multiqubit and multilevel quantum reinforcement learning with quantum technologies", *PLOS ONE*, vol. 13, no. 7, pp. 1–25, Jul. 2018. DOI: 10.1371/journal.pone.0200455. [Online]. Available: https://doi.org/10.1371/journal.pone.0200455.

[13] I. Cong, S. Choi, and M. D. Lukin, "Quantum convolutional neural networks", Oct. 2018. arXiv: 1810.03787v1 [quant-ph]. [Online]. Available: http://arxiv.org/abs/1810.03787v1.

[14] A. Daskin, "A simple quantum neural net with a periodic activation function", Apr. 2018. arXiv: 1804.07633v3 [quant-ph]. [Online]. Available: http://arxiv.org/abs/1804.07633v3.

[15] E. Farhi and H. Neven, "Classification with quantum neural networks on near term processors", Feb. 2018. arXiv: 1802.06002v2 [quant-ph]. [Online]. Available: http://arxiv.org/abs/1802.06002v2.

[16] V. Havlicek, A. D. Córcoles, K. Temme, A. W. Harrow, A. Kandala, J. M. Chow, and J. M. Gambetta, *Supervised learning with quantum enhanced feature spaces*, 2018. eprint: arXiv:1804.11326.

[17] W. Huggins, P. Patel, K. B. Whaley, and E. M. Stoudenmire, "Towards quantum machine learning with tensor networks", Mar. 2018, Quantum Science and Technology, Volume 4, 024001 (2019). DOI: 10.1088/2058-9565/aaea94. arXiv: 1803.11537v2 [quant-ph]. [Online]. Available: http://arxiv.org/abs/1803.11537v2.

[18] A. Ramanan, *Quantum gates and circuits: The crash course*, 2018. [Online]. Available: https://blogs.msdn.microsoft.com/uk_faculty_connection/2018/02/26/quantum-gates-and-circuits-the-crash-course/.

[19] Z. Zhao, A. Pozas-Kerstjens, P. Rebentrost, and P. Wittek, "Bayesian deep learning on a quantum computer", Jun. 2018. arXiv: 1806.11463v2 [quant-ph]. [Online]. Available: http://arxiv.org/abs/1806.11463v2.

[20] *Feature maps - qiskit 0.7 documentation*, https://qiskit.org/documentation/aqua/feature_maps.html, (Accessed on 04/06/2019), 2019.

[21] *Qiskit aqua — algorithms for near-term quantum applications*, https://qiskit.org/aqua, (Accessed on 04/06/2019), 2019.

[22] *Qiskit — quantum information science kit*, https://qiskit.org/, (Accessed on 04/06/2019), 2019.

[23] *Quantum devices & simulators - ibm q*, https://www.research.ibm.com/ibm-q/technology/devices/, (Accessed on 04/06/2019), 2019.