TU WIEN Informatics

Beyond 0's and 1's

Exploring the Impact of Noise, Data Encoding, and Hyperparameter Optimization in Quantum Machine Learning

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Data Science

eingereicht von

Sabrina Herbst, B.Sc.
Matrikelnummer 11807863

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ. Prof. Dr. Ivona Brandić
Mitwirkung: Vincenzo De Maio, Ph.D.

Wien, 16. September 2023

Sabrina Herbst | Ivona Brandić

TU WIEN Informatics

Beyond 0's and 1's

Exploring the Impact of Noise, Data Encoding, and Hyperparameter Optimization in Quantum Machine Learning

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Data Science

by

Sabrina Herbst, B.Sc.
Registration Number 11807863

to the Faculty of Informatics

at the TU Wien

Advisor: Univ. Prof. Dr. Ivona Brandić
Assistance: Vincenzo De Maio, Ph.D.

Vienna, 16th September, 2023

Sabrina Herbst	Ivona Brandić

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Sabrina Herbst, B.Sc.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 16. September 2023

Sabrina Herbst

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisors, Prof. Ivona Brandic and Vincenzo De Maio, for their guidance and commitment throughout the entire research process and for the created opportunities.

I want to thank my sister Natalie for her feedback on early versions of the thesis. Additionally, I extend my heartfelt appreciation to everyone at the HPC lab at TU Wien for the wonderful discussions and occasional distractions from work.

This work has been partially funded through the FFG Flagship project HPQC (High Performance Integrated Quantum Computing) # 897481 and the Standalone Project Transprecise Edge Computing (Triton), Austrian Science Fund (FWF): P 36870- N, 2023. IBM Quantum services were used for this work. The views expressed are personal ones, and do not reflect the official policy or position of IBM or the IBM Quantum team.

Kurzfassung

Da sich die Entwicklung klassischer (von-Neumann) Hardware verlangsamt und moderne *Machine Learning (ML)* Modelle zunehmend mehr Rechenleistung und Speicherplatz benötigen, versuchen Forscherinnen und Forscher neue Wege zu finden, aktuelle und zukünftige Anforderungen zu bewältigen. Neue Post-Moore Architekturen werden stetig weiterentwickelt, und, da sie eine vielversprechende Alternative, unter anderem aufgrund theoretischer algorithmischer *Speedups*, darstellen, hat die Bedeutung von Quantencomputern zugenommen. In diesem Rahmen versucht der Bereich des *Quantum Machine Learning (QML)*, Quantenphänomene zu nutzen, um schnellere Algorithmen und eine bessere Expressivität zu erzielen.

Variational Quantum Algorithms (VQAs) haben aufgrund ihrer Eignung für aktuelle Quantenhardware Aufmerksamkeit erregt. VQAs besitzen jedoch zahlreiche Parameter, einschließlich Datentransformation, Architektur und Training, die die Modelle stark beeinflussen können. Zudem sind aktuelle Quantencomputer anfällig für Fehler, was das Training erschweren und die Leistung der Modelle verringern kann. In dieser Arbeit wollen wir verschiedene Konfigurationen vergleichen und ihre Performance in Gegenwart von ebendiesen Fehlern bewerten, um die effektivsten Ansätze zu ermitteln.

Unsere Experimente zeigen, dass die Wahl des Optimierers und der Datentransformation die Leistung der Modelle erheblich beeinflussen kann. Die Datentransformation allein führt jedoch nicht zu einem leistungsfähigen Modell, sondern muss mit einem guten Ansatz kombiniert werden, um Vorteile gegenüber anderen Konfigurationen zu erzielen. Darüber hinaus verringern Fehler im Quantencomputer die Leistung der Modelle, und beeinflussen insbesondere die besten Modelle.

Abstract

As the development of classical (von-Neumann) hardware is slowing down and state-of-the-art machine learning (ML) models require an ever-increasing amount of computational and storage resources, the research community is tasked with finding ways to cope with current and future demands. New Post-Moore architectures are therefore gaining importance, and, being a promising alternative due to, among others, theoretical algorithmic speed-ups, the significance of quantum computing has been steadily growing. Within this framework, quantum machine learning (QML) seeks to use quantum phenomena to achieve speed-ups and enhanced expressive power.

Among the QML approaches, variational quantum algorithms (VQAs) have attracted attention for their suitability for near-term quantum hardware. However, VQAs require the tuning of various parameters, including data transformation, model architecture, and training, which can significantly impact the models. Additionally, current quantum hardware is subject to noise, which can make training more difficult and reduce the performance of the models. In this study, our objective is to compare different parameter configurations and assess their performance in the presence of noise, with the aim of identifying the most effective settings.

Our experiments show that the choice of optimizer and feature map can significantly impact the performance of the models. The feature map alone does not lead to a well-performing model, rather it needs to be combined with a good ansatz, to lead to advantages over other configurations. Furthermore, we find that noise decreases the performance, impacting the best-performing models the most.

Contents

CHAPTER 1

Introduction

Advances in quantum computing (QC) and the availability of quantum machines in the cloud have made it accessible to researchers of different disciplines [DL21]. Quantum computing is a paradigm of computing that leverages the principles of quantum mechanics. Unlike classical computers, which rely on binary digits (bits) to store and process information, quantum computers use quantum bits (qubits) that can exist in multiple states at once. This allows quantum computers to perform certain computations significantly faster than classical computers and makes completely new and different solutions possible [GI19].

Examples of significantly more efficient algorithms than currently known classical counterparts are Shor's algorithm [Sho97] and Grover's algorithm [Gro96]. Shor's algorithm can factor large numbers in polynomial time, which is an exponential speed-up when compared to the best known algorithm on classical computers, and can, in theory, break RSA encryption ([RSA78]) [GI19]. Grover's algorithm allows searching an unordered list in $O(\sqrt{n})$, as opposed to $O(n)$ of the classical counterpart, thus providing a quadratic speedup [ABBB21].

In recent years, we have experienced an exponential growth of data, and analyzing it can help researchers and organizations extract information from the raw data. Common use cases are fraud detection, object recognition or medical informatics [GCA22]. However, the amount of data is ever-growing and, due to Moore's law slowly coming to an end, researchers hypothesize that classical computers might not be sufficient anymore in the near future [CHI+18, Mar14]. Hence, quantum machine learning (QML) has become an important area of research.

Potentially, QML could offer improvements in terms of runtime and space complexity [Pas23]. In [SK22], the authors mention two directions of research, namely (1) speed-ups, where classical algorithms are translated into more efficient quantum ones and (2) variational quantum algorithms (VQAs), which are models similar to classical

neural networks [ABBB21]. Main research interests for now are related to architectures, gradients, trainability, expressive power and generalization [SK22].

The ultimate goal is to find a *quantum advantage*, i.e. proving that a quantum computer can solve problems that a classical one cannot. For now, due to hardware limitations, experiments are rather limited and can only happen on a small scale (currently, the biggest available IBM quantum computer has 127 qubits[1]). Furthermore, deep learning is highly effective at solving complex problems, due to its ability to automatically learn representations of data through neural networks with multiple layers. Therefore, some argue that small experiments and theoretical proofs will not be able to find this advantage in machine learning for now, but rather should help us understand the methods better [SK22].

1.1 Problem Statement & Research Objective

A big problem in QML is the translation of classical data into a quantum state. Typically, in machine learning, the data encoding is crucial, hence, a lot of thought has to go into initializing the data. Even if QML provides a significant speed-up to the classical counterpart, this advantage should not be cancelled out by the costs of reading the data [MKR+21]. Furthermore, as mentioned in [SK22], similarly to the design of classical neural networks, the architectures of variational quantum circuits crucially affect their performance.

Moreover, in the near future, only Noisy Intermediate-Scale Quantum (NISQ) technology will be available, offering between fifty and a few hundred qubits. As the name suggests, quantum computers are subject to noise [Pre18], which can emerge from various sources, such as unwanted interactions with the environment, other qubits or imperfect operations (e.g., over- or under-rotations) [RK21]. The error rate per two-qubit gates is above 0.1%, and experts are not sure whether such small error rates are even possible on big quantum computers with many qubits [Pre18].

The main goal of this thesis is to take a deeper look into quantum machine learning, in particular variational quantum circuits, and compare different parameter configurations and their susceptibility to noise in the quantum computer. Additionally, we want to compare the results to a classical machine learning model to study how good quantum machine learning models can already perform. In particular, the following research questions will be investigated.

- How do quantum machine learning algorithms perform on standard regression and classification problems in comparison to classical ones?
- How do hyperparameters, such as the ansatz and optimizer, affect the runtime and performance of a variational quantum circuit?

[1] https://www.ibm.com/quantum/roadmap. Accessed 22.06.2023

- How much does data encoding influence performance and runtime of a variational quantum circuit?
- How susceptible is the performance of the ML models to noise in the quantum computer?

We will use appropriate performance measurements for the problem at hand (accuracy and f-1 score for classification, mean squared error (MSE) and mean absolute error (MAE) for regression) and compare the models based on these results. By addressing these research questions, we aim to gain a deeper understanding of the potential and the limitations of QML.

1.2 Expected Contributions

We expect this thesis to shed light on the potential of QML, and highlight how different quantum circuit hyperparameters influence the model. We want to demonstrate how different choices for the main hyperparameters impact the runtime and performance of the models, and provide an extensive evaluation and comparison.

Through the investigation of data encoding techniques, we aim to compare their efficiency and runtime and highlight potential trade-offs between the two. We aim to identify the most suitable approaches for achieving high accuracy in QML algorithms with reasonable time requirements. Due to the noise in current quantum hardware, we want to find out how susceptible the models are. Therefore, we run the algorithms with noise models as well, and evaluate and compare the results to the noiseless ones.

We anticipate that this thesis will contribute to the ongoing effort to bridge the gap between quantum and classical ML by providing a comprehensive evaluation and comparison of the performance of the two on standard regression and classification problems. This could help researchers and practitioners to identify the strengths and weaknesses of both approaches and to determine the most suitable approach for a given problem.

1.3 Outline

The thesis will be structured as follows. In Chapter 2, we will present theoretical aspects of the thesis. In particular, Section 2.1 will give a brief overview on the relevant aspects of quantum mechanics. Section 2.2 will provide an introduction to quantum computing, whereas Section 2.3 will discuss quantum machine learning. We will focus on data encoding and VQAs here.

Then, Chapter 3 will present related work. Section 3.1 will discuss literature and experimental results with respect to data encoding. In Section 3.2, we will give an introduction into recent advances in optimizing VQAs and in Section 3.3, recent literature regarding the ansatz of VQAs will be presented. Then, in Section 3.4 we give an overview

of studies that tried to optimize the hyperparameters of VQAs for machine learning as well.

Moreover, in Chapter 4, we will introduce our methodology. First, in Section 4.1, we will present the four chosen datasets and give an overview of their characteristics. Then, in Section 4.2, we will shortly provide an overview of the models and hyperparameters of our classical ML baseline. Finally, in Section 4.3, we will present our methodology for the QML models.

Then, in Chapter 5, we will present our results. Section 5.1 will discuss the experimental setup and 5.2 will discuss the results from the classical ML baseline. Then, Section 5.3, will present the results of the QML models. In Section 5.4, the results will be compared.

Finally, we will conclude the thesis in Chapter 6 and provide an outlook.

CHAPTER 2

Quantum Computing Fundamentals

Quantum mechanics (QM) is applicable in a variety of disciplines, such as chemistry, physics, molecular biology or cosmology, and the theory has held up through countless experiments. Various technologies, such as transistors, lasers or nuclear reactors, are based on quantum theory, and it is estimated that quantum mechanics are exploited by products that make up about 30% of the US gross national product. It is said to be the most successful quantitative theory ever [Bli21, p.21-34].

The following chapter gives an introduction to the fundamentals of quantum computing (QC). We will initially give an introduction to QM in Section 2.1. In particular, we want to give an overview on the history and why classical mechanics is not sufficient. Furthermore, we will introduce quantum states, superposition, and entanglement using Dirac's mathematical formulation of QM.

Moreover, in Section 2.2, we will give an overview of QC, starting with the motivation and potential advantages. Then, we will write about current quantum hardware and recent advances. Moreover, we give an introduction to quantum computing, including quantum gates, circuits and how useful quantum algorithms can be built.

Finally, in Section 2.3, we will introduce QML. We will point out challenges and potential advantages over classical machine learning. Then, we will introduce the field itself, focussing on variational quantum circuits.

2.1 Quantum Mechanics Basics

Quantum mechanics (QM) is a theory that was formulated based on three phenomena related to electromagnetic radiation that classical physics could not explain. These

phenomena are (1) *blackbody radiation*, (2) the *photoelectric effect*, and (3) the *origin of line spectra*. Starting in about 1900, Max Planck, Albert Einstein and Niels Bohr built on top of each other to find theories explaining the behavior, which ultimately led to the basic theory of QM. The principles of QM were then mainly formulated by Schrödinger, Dirac, and Heisenberg in 1925/1926 [Bli21, p.1-34].

2.1.1 Classical vs. Quantum Mechanics

Determining the appropriate circumstances for applying classical or quantum mechanics is a crucial consideration in understanding the behavior of physical systems. In [Dir58, p.3-4], Dirac describes the difference based on the disturbances an observation causes. In science, objects of observation are explored by manipulating external variables. Therefore, the observable thing is disturbed by measuring it. If the disturbance is negligible, then the object will be termed as *big*, otherwise it will be labelled as *small*. Some level of care must always be taken in order to reduce disruptions when measuring, regardless of the method used. Still, absolute precision (no disturbance at all) cannot be achieved independently of the skill level or technique. Hence, if the inevitable disturbance when observing the object of observation is negligible, it is *big* and classical mechanics apply, else it is *small* and QM is necessary.

These definitions influence causality, which is found in undisturbed objects of observation. When considering and exploring a *small* system, the disturbance is significant, thus, the observations are not necessarily causal. Therefore, observations are themselves not deterministic, but quantum theory gives the means to calculate probabilities of different outcomes [Dir58, p.3-4].

2.1.2 Superposition

The indeterminacy of QM leads to a more complex theory, compared to classical mechanics alone, which is not necessarily desirable. The principle of *superposition* compensates this added complexity. In an atomic system made up of several particles, *states* are all possible motions of the particles in such systems. There are connections between different states, that is, if we know the system to be in one state, it is always in part in two or more other states as well. The state we know the system to be in is a superposition of the other states. Just as every state is a superposition of other states, every two or more states can be superposed into another state [Dir58, p.11-12].

In particular, a superposition shares characteristics with the states making it up and is something between them. Depending on the weight of the states, it is more similar to one or the other and there is a higher probability of measuring one, if its weight is higher. The result of a measurement is either one state or the other, but the probability of measuring one or the other is influenced by the relative weights of the states in the superposition. [Dir58, p.13].

Although attempts have been made to compare this principle to vibrating strings or membranes in classical mechanics, such analogies are misleading because classical me-

chanics is deterministic, whereas QM is non deterministic. As Dirac notes in [Dir58, p.14], the principle of superposition is a unique feature of QM that cannot be explained by classical mechanics.

One of the most famous examples of a superposition is the paradox of Schrödinger's cat. In this thought experiment, a cat is placed in a sealed container with a device that could release enough poison to kill the cat, which is triggered by a quantum mechanical measurement. The two possible states in superposition are the cat being dead or alive, and the overall state of the cat is not in either of them, but rather in a superposition of them. However, once we look into the container, it is either dead or alive. The paradox arises because the cat appears to be both dead and alive at the same time, which contradicts common sense. For further information on the experiment, please refer to [Vil86] for a detailed explanation and discussion.

2.1.3 The Uncertainty Principle

One of the most influential theories in QM is Heisenberg's uncertainty principle. The following notation will be used: q refers to the position of a particle, and p refers to the momentum in the classical sense. Additionally, q' and p' denote quantum mechanical observables that can only be measured probabilistically, with $\Delta p'$ and $\Delta q'$ representing the uncertainties in measurement. The constant h is known as Planck's constant [Dir58, p.97-99].

$$\Delta q' \Delta p' = h \tag{2.1}$$

Heisenberg's Principle of Uncertainty is defined in Equation 2.1. It follows that the more certain one variable can be observed, the less certain one can be about the second variable. Therefore, if either q' or p' are certain, the other one is completely unknown, i.e., all possible measurements have equal probability. This influences the view of a system, because it implies that we can never know all details of it. Furthermore, it influences *quantum entanglement* [Dir58, p.97-99].

2.1.4 Entanglement

Another interesting phenomenon in QM is *entanglement*. The original thought experiment, called the Einstein-Podolsky-Rosen experiment [EPR35], considers two particles from a common source that move into opposite directions triggered by some event. The position of the first particle and the momentum of the second particle are measured at the same time. As the conservation of momentum (total momentum in a closed system remains constant) is also valid in QM, the momentum of the first particle equals the negative momentum of the second particle. Hence, it is possible to know the position and momentum of the first particle at the same time, which contradicts the uncertainty principle [Bli21, p.342-345].

To overcome this inconsistency, it is suggested that when measuring the particles simultaneously, one must take into account their correlation due to them being part of an indivisible quantum system. In simpler terms, measuring the position of the first particle directly impacts the momentum of the second one. Therefore, the momentum of the first particle is affected by this interaction, and given their dependency, the particles are considered to be *entangled*. [Bli21, p.342-345].

Entanglement between particles is not dependent on their distance from each other, meaning that even if the particles were separated by light-years, measuring one particle would directly affect the other. This implies that communication (in this case, the disturbance caused by measurement) could theoretically occur faster than the speed of light, which would contradict the principle of locality in Einstein's theory of relativity. According to Bell's Theorem, there is a fundamental incompatibility between the concept of local relativity and quantum entanglement. Experiments have since confirmed this prediction by demonstrating that our universe does not adhere to local relativity [Bli21, p.342-348].

2.1.5 Mathematical Formulation

Paul Dirac introduced the widely used notation in QM known as the *bra-ket notation*. The following information on the mathematical notation is based primarily on Sections 5 and 6 (pages 14-22) of his book [Dir58], unless otherwise noted. All equations referenced can be found in the book.

The definition of states is an important starting point. One could argue superposition to be a summation of two states to form a new one. Therefore, if one would want to come up with a mathematical system, it should allow for such operations. Vectors are not sufficient, as they only exist in finite dimensions. Dirac came up with his own notation, calling them *ket vectors*. They are more powerful than classical vectors, as they allow infinite dimensions. A ket vector A is denoted by $|A\rangle$. It is possible to multiply them with complex numbers, to add two or more ket vectors together or even integrate them.

The *state* of a system at a given point in time is represented by one ket vector. If a new state is formed by a superposition of two other ones, it is possible to represent it linearly with respect to the original two states, as can be seen in Equation 2.2. The two ket vectors $|A\rangle$ and $|B\rangle$ are multiplied by a complex number each and added together (forming a superposition) to create $|R\rangle$.

$$c_1|A\rangle + c_2|B\rangle = |R\rangle, c_1, c_2 \in \mathbb{C} \tag{2.2}$$

The ket vector $|R\rangle$ is the same, independently of the order of $|A\rangle$ and $|B\rangle$. Equation 2.2 shows that it is also possible to use a superposition of $|B\rangle$ and $|R\rangle$ to get to $|A\rangle$, meaning the superposition relationship is symmetrical between the kets.

Furthermore, we look at Equation 2.2, to investigate how the state $|R\rangle$ changes with c_1 and c_2. Multiplying them with the same number yields the same state $|R\rangle$, only the ratio

of c_1 and c_2 influences the state. The ratio also influences the probability of measuring one state or the other.

The superposition of a state with itself is the same state, as can be seen in Equation 2.3. Hence, a state is defined only by the direction, not by the length. The direction of the state carries information about physical properties of the system, whereas the length does not. This stands in contrast to classical mechanics, i.e., a vibrating membrane, where superposition with itself leads to different oscillations, hence, a different state.

$$c_1|A\rangle + c_2|A\rangle = (c_1 + c_2)|A\rangle, c_1, c_2 \in \mathbb{C}\backslash\{0\} \tag{2.3}$$

There exists a dual vector to the ket vector, called the *bra vector*. In particular, for every ket vector, there exists a number ϕ, which is a linear function of the ket. The number can be found by calculating the scalar product between the ket vector $|A\rangle$ and the corresponding bra vector $\langle A|$, written as $\langle A|A\rangle$. The two vectors are called *conjugate imaginary* of each other. The state of a system can be defined by both, the bra or the ket vector, as there is a one-one correspondence between the two.

2.2 Quantum Computing Principles

In this section, we will provide an introduction to quantum computing. First, we will explore the motivation and potential advantages of quantum computing in Section 2.2.1. Next, we will discuss the current state of quantum hardware and challenges in Section 2.2.2. While quantum computing is still in its infancy, there has been rapid progress in the development of quantum hardware, with several companies and research organizations working to build quantum computers with increasingly larger numbers of qubits.

Afterwards, we will introduce the fundamental principles of quantum computing, starting with quantum bits in Section 2.2.3. This will include an explanation of qubits, the basic building blocks of quantum computers, and their unique properties. We will discuss quantum gates, which operate on qubits, and quantum circuits afterward, in Section 2.2.4.

2.2.1 Motivation

The idea of quantum computers originated quite early, with the physicist Richard Feynman already talking about its potential in 1982 [Bli21, p.355-378]. David Deutsch published a paper in 1985 writing that quantum computers are a generalization of Turing machines, that they can be created and that they have the potential to create fundamentally different results due to their unique characteristics [Deu85].

Early on, researchers recognized the immense potential of the phenomena of superposition and entanglement in computing, as they can lead to a significant increase in efficiency [Bli21, p.355-378]. Many researchers believe that quantum computers are more

powerful than classical computers for several reasons. Notably, quantum computers can solve certain problems that are difficult for classical computers, such as factoring large numbers using Shor's algorithm. Additionally, there exist quantum states that are extremely challenging to replicate using classical methods, including the ability to sample from distributions that are difficult to even approximate classically ([LBR17]) [Pre18].

Beyond classical algorithms, machine learning is another interesting area of research and the center of this thesis. As the amount of data is ever-growing and having recently entered the Post-Moore era, researchers hypothesize that classical computers might not be sufficient anymore in the near future [CHI+18, Mar14]. Potentially, quantum computers could offer improvements in terms of runtime and space complexity.

However, even beyond computer science, there are a lot of scientific disciplines that could benefit from quantum computers. The theoretical physicist John Preskill describes being fascinated by QC, as it offers the potential to simulate any processes in nature. In particular, processes, such as the formation of the universe after the big bang, could be simulated, allowing new insights into fundamental physics [Pre18].

2.2.2 Hardware and Near-Future Expectations

The field of quantum computing is rapidly advancing, and Tech companies are heavily investing. According to McKinsey, 1.4 billion USD were invested in start-ups in 2021 and the market estimate for 2040 is at 90 billion USD[1]. Among the most prominent companies developing quantum computers are IBM[2], Google[3] and Microsoft[4].

One of the biggest challenges in QC arises from the definition of quantum mechanics (QM) itself. In Section 2.1, the concept of a significant disturbance that happens when observing a quantum system was explained. Hence, it is necessary to keep the system as isolated as possible, while keeping the possibility to interact with the system. Furthermore, quantum bits (qubits), the quantum computing counterparts of classical bits, have to interact with each other, and it needs to be possible to read out the result in the end. These requirements are quite restrictive and partly contradictory [Pre18].

With regard to these difficulties, we mentioned the terminology of a Noisy Intermediate-Scale Quantum (NISQ) computer earlier. It refers to a quantum computer with fifty to a few hundred qubit with significant levels of noise. Although there is a lot of effort going into quantum error correction, ways to correct the effects from the noise, there is a big overhead associated with it, and it is not possible to sufficiently mitigate the consequences of noise yet [Pre18].

[1] https://www.mckinsey.com/featured-insights/themes/how-quantum-computing-could-change-the-world. Accessed: 13.04.2023

[2] https://www.ibm.com/quantum. Accessed: 13.04.2023

[3] https://quantumai.google/. Accessed 13.04.2023

[4] https://azure.microsoft.com/en-us/solutions/quantum-computing/. Accessed: 13.04.2023

With respect to noise, one can roughly distinguish four different types in a quantum computer, which are explained in the following and are listed in [RK21].

1. *Environment*: as already described earlier, it is not possible to keep a quantum computer perfectly isolated as, in the least, we need to be able to execute commands and retrieve the results.
2. *Other qubits*: entanglement is among the most important concepts in QC. However, it is not possible to perfectly control which qubits are entangled. From time to time, qubits may become entangled accidentally, leading to unwanted side effects.
3. *Operations*: operations are not guaranteed to be done perfectly, hence, small errors (slight over or under rotations) can significantly impact the results. These errors are among the most common.
4. *Leakage*: quantum computers do not necessarily only allow two states, rather two of the possible states are defined to be $|0\rangle$ and $|1\rangle$ (which will be introduced later). Should a qubit enter a different state unintentionally, it is called leakage.

Hence, there are several challenging aspects to building quantum systems. However, the field is rapidly advancing. For example, IBM released their first quantum computer with only five qubits in 2016[5]. In 2019, they released their first 27 qubit quantum computer Falcon and only presented their 433 qubit quantum computer Osprey in 2022. In 2023, they are planning to release their first 1121 qubit machine Condor. According to their roadmap[6], they plan to scale their quantum systems to tens or hundreds of thousands qubits by 2026, by joining multi-chip processors (so-far they have used only single-chip processors) and quantum communication technologies.

2.2.3 Quantum Bits

A quantum computer is, essentially, a very simple physical system, containing a finite number of smaller quantum-mechanical two-state systems (quantum bits, qubits) [Mer07, p.xii]. In a classical sense, a bit can be in two orthogonal states (orthogonal being defined equivalently as for vectors), namely $|0\rangle$ and $|1\rangle$, shown in Equation 2.4. In the case of two bits, they are in one of four states ($|0\rangle|0\rangle = |00\rangle, |01\rangle, |10\rangle, |11\rangle$), the states forming an orthonormal basis. In general, n bits span a vector space of 2^n dimensions [Mer07, p.6]

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{2.4}$$

The notation already implies that the bits are multiplied, in particular, the *tensor product* $\otimes$ is used. The tensor product of vectors $a \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$ is a vector of size $N * M$

[5] https://research.ibm.com/blog/quantum-five-years. Accessed: 13.04.2023
[6] https://www.ibm.com/quantum/roadmap. Accessed: 13.04.2023

containing the product of all possible combinations of the two vectors (i.e. element (γ, η) will return the product $a_\gamma b_\eta$) [Mer07, p.6]. An example can be seen in Equation 2.5.

$$\begin{pmatrix} 2 \\ 3 \\ 4 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 2*1 \\ 2*2 \\ 3*1 \\ 3*2 \\ 4*1 \\ 4*2 \end{pmatrix} = \begin{pmatrix} 2 \\ 4 \\ 3 \\ 6 \\ 4 \\ 8 \end{pmatrix} \tag{2.5}$$

The state of a **qubit** is defined as a normalized superposition of the base states, as can be seen in Equation 2.6, subject to the condition shown in Equation 2.7. Geometrically speaking, the state of a qubit is in the vector space that is spanned by the orthogonal base vectors $|0\rangle$ and $|1\rangle$. α_0 and α_1 are called the *amplitude* of the state. A qubit does not have the value $|0\rangle$ or $|1\rangle$, rather, qubits are *described* by or *associated* with states [Mer07, p.17-19].

$$|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle = \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix}, \alpha_0, \alpha_1 \in \mathbb{C} \tag{2.6}$$

$$|\alpha_0|^2 + |\alpha_1|^2 = 1, \tag{2.7}$$

Similarly, the state of two qubits is defined in Equation 2.8 (the amplitude has to be normalized) and the state made up of n qubits can be seen in Equation 2.9. The set of possible states ($0 \leq x < 2^n$) is called the *computational basis*. The subscript n here denotes the number of qubits in the computational basis state. Combining two states is done by taking the tensor product [Mer07, p.17-19].

$$|\Psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle \tag{2.8}$$

$$|\Psi\rangle = \sum_{0 \leq x < 2^n} a_x |x\rangle_n, \sum_{0 \leq x < 2^n} |a_x|^2 = 1, n \in \mathbb{N} \tag{2.9}$$

The biggest difference between classical bits and qubits is that classical bits are always in one of the 2^n computational basis states and can be expressed by the individual bits. This is not true for states of two or more qubits, as they are in a superposition, where it is not necessarily possible anymore to associate a state with a single qubit. If it is not possible to describe individual qubits of a state independently of other qubits, the qubits are called *entangled* [Mer07, p.17-19]. Entanglement and superposition are often a fundamental building block of efficient quantum algorithms (i.e. Grover's algorithm [Gro96]) [ABBB21].

2.2.4 Operations on Qubits

Quantum computers operate on qubits by performing *linear*, *reversible* operations, meaning the operation can be undone again. There are two reversible operations on a single qubit, leaving it as-is or flipping it. The not operator X shown in Equation 2.10 turns $|0\rangle$ into $|1\rangle$ and vice versa, whereas the identity operator I, shown in Equation 2.11, leaves it as-is. The operators are usually represented as matrices, which are multiplied with the state vectors [Mer07, p.8-11].

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{2.10}$$

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{2.11}$$

One of the simplest operations on multiple qubits is swapping two of them, using the Swap gate $S_{i,j}$. The controlled NOT (cNOT) operator $C_{i,j}$ performs a not operation on qubit j if qubit i is in state $|1\rangle$ and it is among the most fundamental operators in QC. A lot of two-qubit gates can be constructed out of cNOT gates. For example, one can construct the $S_{i,j}$ gate from three cNOT gates, as $S_{i,j} = C_{i,j}C_{j,i}C_{i,j}$ [Mer07, p.8-11].

Further important gates are the Z gate, shown in Equation 2.12, and the Hadamard transformation H, shown in Equation 2.13 [Mer07, p.13].

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{2.12}$$

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{2.13}$$

It is essential, after having performed an operation on a state, to end up with a unit vector again. Such operations are called *unitary*. Furthermore, unitary operations always have a unitary inverse (i.e., reversing an operation always results in a unit vector as well) [Mer07, p.19].

Unitary transformations on one qubit are called 1-qubit gates, those that act on two are called 2-qubit gates. Physically building qubit gates grows more difficult with an increasing number of qubits. Therefore, the focus is on 1- and 2-qubit gates, which are already a challenge to build reliably. However, this is not a limitation. Any unitary transformation on a quantum computer can be represented by 1- and 2-qubit gates. This has been proven in [DiV95]. For a concise proof, please refer to the paper. For classical computers, 3-bit gates are required [Mer07, p.20]

While one can always look at the values of classical bits to read them without a problem, it is never possible to read the state of a quantum computer. Hence, it is not possible

to read the amplitude of a state. One can extract information from qubits by making a measurement, which is the only irreversible operation one can make, where each qubit will take on either $|0\rangle$ or $|1\rangle$ [Mer07, p.23-24].

The probability of what we measure was figured out by Max Born and is called the Born Rule, In particular, the amplitude is representative of the probability of measuring $|0\rangle$ or $|1\rangle$. State $|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$ will have a probability of $|\alpha_0|^2$ of measuring $|0\rangle$ and a probability of $|\alpha_1|^2$ of measuring $|1\rangle$. Generally speaking, if we measure state $|\Psi\rangle$ in Equation 2.14, the probability of measuring the integer x is given by Equation 2.15. [Mer07, p.23-24]

$$|\Psi\rangle = \sum_{0 \leq x < 2^n} \alpha_x |x\rangle_n \tag{2.14}$$

$$p(x) = |\alpha_x|^2 \tag{2.15}$$

Born's rule should not be misinterpreted as the 'real' state of a qubit being in $|0\rangle$ with probability $|\alpha_0|^2$ and vice versa. The qubit does not carry a value beforehand, it only takes on a value when measuring it [Mer07, p.26-27]. This works analogously to Schrödinger's cat not being dead or alive before we look at it, rather it is something in-between (a superposition). The following analogy describes well how one should interpret a measurement.

> "While measurement in quantum mechanics is not at all like measuring somebody's weight, it does have some resemblance to measuring Alice's IQ, which one can argue, reveals no preexisting numerical property of Alice, but only what happens when she is subjected to an IQ test." [Mer07, p.25]

Measuring is done using a n-qubit measurement gate. As stated earlier, the operation is not reversible, once measured, the set of qubits take on the measured value, and it is not possible to reconstruct the state again [Mer07, p.25]. An excellent resort would therefore be to clone the state beforehand. Unfortunately, this has been proven to be impossible in [WZ82]. For a concise proof, please refer to the original paper. In other words, by measuring the state of n qubits, it becomes impossible to reconstruct the state again. The process is called *reduction* or *collapsing* of the state [Mer07, p.26]

The computational process therefore seems pretty restrictive. In order to retrieve reasonable results from quantum computers, the goal is usually to end up in a state where most of the amplitudes are (close to) zero, and only one is not. With a high probability, one retrieves the desired result [Mer07, p.26].

It is not possible to know the state of qubits or potential entanglements with other qubits prior to starting off with an algorithm without measuring it. Therefore, measurement plays an important role in the beginning as well. The process of measuring and transforming the desired qubits (usually into $|0\rangle_n$) before using them is called *state preparation*. The

process consists of (1) measuring them and (2) applying an X gate to all qubits in state $|1\rangle$ [Mer07, p.30-31].

Finally, we want to introduce the term *quantum circuit*, referring to a set of operations on qubits. In particular, a circuit uses quantum gates to manipulate the state of qubits [Bli21, p.358].

2.2.5 Quantum Algorithms

To conclude this section, we want to provide an idea of how quantum algorithms can actually be built, given that the previously defined constraints seem to be quite restrictive.

Beforehand, however, we want to clarify notations. When looking at quantum algorithms, we usually consider a set of qubits for the input (*input register*) and one for the output (*output register*). The output register is required to make sure everything is reversible, if an algorithm assigns different inputs to the same output. Additional qubits are of course possible too.

Now, Equation 2.16 shows the notation, where $|x\rangle_n$ is the input register, $|y\rangle_m$ is the output register and $f(x)$ the function applied. U_f is the unitary transformation in which we apply $f(x)$. The symbol $\otimes$ is a bitwise exclusive or (XOR). If the initial state of the output register is $|0\rangle_m$, it is easy to see that $|y \oplus f(x)\rangle_m = |f(x)\rangle_m$. Additionally, note that $|x\rangle_n$ stays as-is [Mer07, p.36-40].

$$U_f(|x\rangle_n|y\rangle_m) = |x\rangle_n|y \oplus f(x)\rangle_m \tag{2.16}$$

We will introduce a simple quantum algorithm here to showcase how we can use quantum computers to solve problems more efficiently. In particular, we will look at Deutsch's problem, which was first introduced by David Deutsch in [DJ92]. It was one of the first proposed problems that could be much more efficiently solved on quantum computers. However, in the first proposal, the answer was correct only half the time. It was efficiently and properly solved in [CEMM98].

In particular, we are looking at a function $f : \{0, 1\} \to \{0, 1\}$, for which, trivially, only exist four different realizations. The same function as in Equation 2.16 is given, i.e. one can evaluate the function but does not explicitly know which of the four functions $f(x)$ it is. Classically, to find out the function, one would run $U_f(|0\rangle|0\rangle)$ and $U_f(|1\rangle|0\rangle)$ to find the answer, but for Deutsch's problem, the function can be invoked only once [Mer07, p.41-46].

Now, we want to find out whether the underlying function f is constant ($f(0) = f(1)$). It is easy to see that the only way to figure this out is to run the algorithm twice. On a quantum computer, however, we only need one run, which we will explain shortly. In the quantum algorithm, however, we lose the possibilities of getting the values $f(0)$ and $f(1)$ themselves, we only look at the relation between them. Interestingly, this is the QC realization of the Uncertainty Principle (see Section 2.1.3) [Mer07, p.40-46].

To solve the problem, we create a superposition of qubits in the beginning. In particular, the Hadamard transformation (shown in Equation 2.13) can be used for this purpose, as shown in Equation 2.17. We apply a Hadamard transformation to the n input qubits (the transformation is a tensor product of n Hadamard transformations) and an identity transformation to the output qubits. We now have a superposition of all possible computational basis states [Mer07, p.36-40].

$$\begin{aligned} U_f(H^{\otimes n} \otimes 1_m)(|0\rangle_n|0\rangle_m) &= \frac{1}{2^{n/2}} \sum_{0 \leq x < 2^n} U_f(|x\rangle_n|0\rangle_m) \\ &= \frac{1}{2^{n/2}} \sum_{0 \leq x < 2^n} |x\rangle_n|f(x)\rangle_m \end{aligned} \tag{2.17}$$

Hence, the state of the whole input registry is now defined by the result of all 2^n function evaluations, which is termed *quantum parallelism*. However, this does not mean it is possible to read out all the function evaluations, as it is never possible to know the state. If we measured the state at this point, we will get one random (all states have the same probability) state [Mer07, p.36-40].

For Deutsch's problem, both the input and the output registry have only one qubit. Applying the Hadamard transformation to the input registry yields Equation 2.18. Measuring this state gives the result of $f(0)$ with a 50% probability and $f(1)$ with a probability of 50%. However, as noted before, we are not interested in the exact values themselves [Mer07, p.41-46].

$$U_f(H \otimes 1)(|0\rangle|0\rangle) = \frac{1}{\sqrt{2}}|0\rangle|f(0)\rangle + \frac{1}{\sqrt{2}}|1\rangle|f(1)\rangle \tag{2.18}$$

Applying first NOT (X) and then Hadamard (H) gates to both registers, yields Equation 2.19.

$$\begin{aligned} (H \otimes H)(X \otimes X)(|0\rangle|0\rangle) &= (H \otimes H)(|1\rangle|1\rangle) \\ &= (\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle)(\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle) \\ &= \frac{1}{2}(|0\rangle|0\rangle - |1\rangle|0\rangle - |0\rangle|1\rangle + |1\rangle|1\rangle) \end{aligned} \tag{2.19}$$

When applying the unitary transformation U_f, we can use the linearity to arrive at Equation 2.20. To explain the second line, we define $\tilde{x} = 1 \oplus x$, i.e., if $x = 1$, the XOR will yield 0, else 1. We do the same for our function $f(\tilde{x}) = 1 \oplus f(x)$. Note that $|1 \oplus f(x)\rangle$ is simply the inverse of $|0 \oplus f(x)\rangle$ (from the definition of the XOR) [Mer07, p.41-46].

$$\begin{aligned}&\frac{1}{2}(U_f(|0\rangle|0\rangle) - U_f(|1\rangle|0\rangle) - U_f(|0\rangle|1\rangle) + U_f(|1\rangle|1\rangle)) \\ &= \frac{1}{2}(|0\rangle|f(0)\rangle - |1\rangle|f(1)\rangle - |0\rangle|f(\tilde{0})\rangle + |1\rangle|f(\tilde{1})\rangle)\end{aligned} \tag{2.20}$$

If $f(x)$ is constant, $f(0) = f(1)$ and $f(\tilde{0}) = f(\tilde{1})$, if it is not, $f(\tilde{1}) = f(0)$ and $f(\tilde{0}) = f(1)$. If the function is constant, Equation 2.20 reduces to Equation 2.21, otherwise, it reduces to Equation 2.22 [Mer07, p.41-46].

$$\begin{aligned}&\frac{1}{2}(|0\rangle|f(0)\rangle - |1\rangle|f(0)\rangle - |0\rangle|f(\tilde{0})\rangle + |1\rangle|f(\tilde{0})\rangle) \\ &= \frac{1}{2}(|0\rangle - |1\rangle)(|f(0)\rangle - |f(\tilde{0})\rangle)\end{aligned} \tag{2.21}$$

$$\begin{aligned}&\frac{1}{2}(|0\rangle|f(0)\rangle - |1\rangle|f(\tilde{0})\rangle - |0\rangle|f(\tilde{0})\rangle + |1\rangle|f(0)\rangle) \\ &= \frac{1}{2}(|0\rangle + |1\rangle)(|f(0)\rangle - |f(\tilde{0})\rangle)\end{aligned} \tag{2.22}$$

Furthermore, we can apply more transformations after the function, for example another Hadamard transformation to the input registry, which will suffice. The final equation is shown in Equation 2.23 (we show how to get there from Equation 2.22 in Equation A.1 in Appendix A). Therefore, by measuring the input register, it is possible to answer the question. The remarkable thing here is, that we learn nothing about the function itself (we still do not know which function underlies the algorithm), we just learn about the relation between $f(0)$ and $f(1)$ [Mer07, p.41-46].

$$\begin{aligned}&(H \otimes 1)U_f(H \otimes H)(X \otimes X)(|0\rangle|0\rangle) \\ &= \begin{cases} f(0) = f(1), |1\rangle\frac{1}{\sqrt{2}}(|f(0)\rangle - |f(\tilde{0})\rangle \\ f(0) \neq f(1), |0\rangle\frac{1}{\sqrt{2}}(|f(0)\rangle - |f(\tilde{0})\rangle \end{cases}\end{aligned} \tag{2.23}$$

2.3 Quantum Machine Learning

Finally, in this section, we want to give an overview of quantum machine learning (QML). We will start by giving an introduction to the potential and promises of QML. Afterward, we will quickly give an overview of machine learning (ML). Furthermore, we will present different data encoding techniques and, finally, introduce variational quantum algorithms (VQAs).

Algorithm	Quantum Tool	Quantum Speedup
PCA	Phase estimation	Exponential
K-Medians	Grover + Qdist	Quadratic
Hierarchical clustering	Grover + Qdist	Quadratic
K-Means	Qdist	Exponential
Several classifiers	Qdist	Exponential
SVM	HHL algorithm	Exponential
Neural Networks	Variational Quantum Algorithms	?

Table 2.1: Overview QML Speedup [Pas23, p.4]

Table 2.1 gives a quick overview of ML algorithms, quantum tools used to implement them on quantum hardware and the speedup obtained. The original table can be found in [Pas23, p.4]. In particular, it shows that QC is able to obtain significant speedups when compared to classical ML.

We will not dive deep into the quantum tools, except for VQAs, but phase estimation [Kit95] is used to calculate eigendecompositions and Qdist [Pas23, p.54-56] is used to calculate Euclidean distances efficiently. The HHL algorithm [HHL09] can solve linear systems of equations efficiently. Due to the discrepancy between theory and empirical evidence of classical neural networks, the advantage for them is not clear yet [Pas23, p.5].

2.3.1 Potential and Promises

Although the field of QML only really took off in the past years due to advances in quantum computers, the first QML algorithms were already published twenty years ago [Pas23, p.1]. In 2003 an algorithm for pattern recognition was proposed. Using the Quantum Fourier Transformation, it obtained an exponential speed-up compared to the classical counterpart [Sch03].

One of the main motivations behind QML is that, although the computational power of classical computers increased drastically (see Moore's law [Sha20]), the improvement is slowing down. Current ML models are very computationally intense and state-of-the-art models train days or months, i.e., Meta's biggest LLAMA model took 1,022,362 GPU-hours to train [TLI+23]. Therefore, alternatives may be necessary in the near future [Pas23, p. 2-4].

Still, the goal of quantum computing for ML goes well beyond speed-ups. There are possible quantum advantages to be found in space efficiency, performance, and expressiveness of models. Similarly to classical ML, neural networks (NNs) are of particular interest. However, due to the gap between classial NN theory and experimental results, it is quite difficult to prove any quantum advantages [Pas23, p.4-5].

It is often argued that simply showing that quantum computers can enhance the field of machine learning might not be the best goal for QML. Rather, it should be seen as a new paradigm, not only influencing the models themselves, but also data representation

and feature encoding. That way, QML can even influence ML, i.e., with the so-called quantum-inspired machine learning (see [SGF19]) [Pas23, p.vii].

2.3.2 Machine Learning

The goal of ML is to use data to make computers find patterns in phenomena. The field has existed for quite some time, but more powerful computers have accelerated the field drastically in recent times [Pas23, p.2-4].

In general, one distinguishes between three different ML types, supervised, unsupervised and reinforcement learning. The interested reader is referred to [Pas23] for an explanation of the three. While there already exist quantum algorithms for all three, the experiments in this thesis are based on supervised learning, where we have a target value for every sample and try to find a ML model describing their relationship. [Pas23, p.2-4].

We therefore define a data point as a vector of numbers (each being called a feature). In particular, a data point of a dataset with d features lies in $\mathbb{R}^d$. In our case, each data point x_i is associated with a label $y_i \in \mathbb{R}$, which we want to capture in our model [Pas23, p.2-4].

2.3.3 Data Encoding

Working with classical data on quantum computers requires an effective strategy for encoding this information into a quantum state. In general, superposition and entanglement are powerful tools that can be leveraged to achieve higher efficiency [Pas23, p.25-28]. Several approaches have been proposed, and we will dive into the most important ones in this section.

As usual in machine learning (ML), data encoding plays a crucial part for the performance of the models. In particular, in [SSM21] the authors show that, essentially, independently of the architecture of a quantum circuit, the data encoding defines and may restrict the functions the circuit is able to express afterward. More specifically, they show that a quantum model with one variable can be represented as a partial Fourier series. The data encoding defines the frequencies it is able to approximate, whereas the architecture only affects the coefficients. Hence, data encoding asymptotically allowing a sufficient number of frequency encodings makes quantum circuits universal function approximators.

Moreover, quantum computing has frequently been compared to kernel methods in classical ML. There are some non-negligible parallels between the two, which are worth highlighting. Specifically, in the course of data encoding, the classical data is mapped into a highly-dimensional Hilbert space. The objective is to obtain a representation that enables efficient computation of the intended task [SK19].

A quantum feature encoding should possess multiple desirable characteristics. Initially, the number of gates used should increase linearly with the number of qubits. The encoding process should be bijective, meaning that each unique feature vector must correspond to a unique quantum state. Additionally, the circuit's depth should be

subpolynomial, and the entire transformation should be efficient in terms of hardware. In this context, hardware efficiency refers to the extra resources required by the gates used in the transformation. [LC20].

The following three approaches are commonly used, and we will introduce them shortly to give an intuition about data encoding. Afterward, we will introduce a feature map based encoding with desirable properties, which is commonly used in QML.

Basis Encoding We consider a set of samples X of length $|X|$. Essentially, we convert all samples $x \in X$ into the orthonormal basis states of a Hilbert space of dimension $|X|$. These orthonormal basis states, represented as $\{|x\rangle\}_{x \in X}$, serve as computational basis states. A classical bit can be transformed using $0 \rightarrow |0\rangle$ and $1 \rightarrow |1\rangle$. Hence, by using n qubits, a n-bit string can be encoded as an orthonormal basis to represent a data point. The Hilbert space now has a dimension of 2^n. Through superposition, one can exploit quantum parallelism, which is used in almost all quantum algorithms, such as Grover's algorithm [Gro96]. Nonetheless, there exist more space-efficient methods to encode classical data into a quantum state. [Pas23, p.25-26].

Amplitude Encoding Instead of using the computational basis states, it is possible to encode the data into the amplitudes of the qubits. We now represent the data point as a complex vector of size d ($x \in \mathbb{C}^d$), and the Hilbert space $\mathcal{H}$ has dimensionality d. The quantum state is $|\psi_x\rangle = \sum_{i \in \{1,\dots,d\}} x_i |\phi_i\rangle$. $|\phi_i\rangle$ signifies a computational basis in $\mathcal{H}$. From the definition of a qubit follows that our vector x needs to be normalized ($\sum_{i \in d} x_i^2 = 1$). The significant advantage when compared to basis encoding is that we can encode complex vectors of size 2^n, instead to n classical bits [Pas23, p.25-26].

Angle Encoding Angle encoding is characterized by a constant depth and is computed as follows: $|x\rangle = \otimes_{i=1}^{N} cos(x_i)|0\rangle + sin(x_i)|1\rangle$. It can be observed that we need n qubits to encode n features. However, there exists another alternative known called dense angle encoding, which employs one qubit for two features, which is expressed as $|x\rangle = \otimes_{i=1}^{\lceil N/2 \rceil} cos(\pi x_{2i-1})|0\rangle + e^{2\pi i x_{2i-1}} sin(\pi x_{2i-1})|1\rangle$. This encoding technique can also be extended to work for any two functions [LC20].

To eventually get to a quantum advantage, it is favorable to choose a feature representation that is hard to obtain or simulate on a classical computer. Therefore, the main objectives are to find a mapping that is (1) useful for ML tasks and (2) difficult to get or approximate classically. Based on this, it is even more interesting to study how expressive such mappings are and can be in compared to classical ML [GTN21].

In [HCT+19], the authors propose a feature mapping that exploits the dimensionality of the Hilbert space, as shown in Equation 2.25, where the Z transformation (Equation 2.24) is used as the Pauli matrix. Using X, Y, Z and the identity I are all viable options for the Pauli matrices.

n refers to the number of features, and S describes the connections between the qubits with $S \in \binom{n}{k}$, with $k \in 1 \dots n$. The connections are influenced by the entanglement parameter and the Pauli matrices. $\phi_S(x)$ refers to a non-linear function. A commonly used non-linear function can be seen in Equation 2.26. The feature mapping can be applied to the input registry $|0\rangle_n$ an arbitrary number of times, adding Hadamard gates in-between. An example with two repetitions can be seen in Equation 2.27.

The authors in [HCT+19] expect the feature map to be difficult to estimate classically, even with a depth of only two. They base their argument on the similarity of the feature map to the circuit of another problem [BLSF19]. For further information, please refer to [HCT+19]. Due to its desirable properties, this feature mapping is widely used for QML, and we will also employ it in our experiments.

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{2.24}$$

$$U_{\psi(x)} = exp(i \sum_{S \subseteq [n]} \phi_S(x) \prod_{i \in S} Z_i) \tag{2.25}$$

$$\phi_S(x) = \begin{cases} x_0, \text{if k} = 1 \\ \prod_{j \in S}(\pi - x_j), \text{else} \end{cases} \tag{2.26}$$

$$|\psi(x)\rangle = U_{\psi(x)} H^{\otimes n} U_{\psi(x)} H^{\otimes n} |0\rangle_n \tag{2.27}$$

2.3.4 Variational Quantum Algorithms

In the following, we will introduce variational quantum algorithms (VQAs). Specifically, we will first introduce the general notion of a VQA and introduce advantages and challenges. Afterward, we will give an overview on ansatzes, in particular, the ones we will use in our experiments. Finally, we will provide an introduction to optimization in VQAs.

In the beginning, we want to remark that VQAs are known under a lot of different names. When reading QML literature, the reader may come across many terms. Such include parametrized quantum circuits, quantum circuit learning, quantum neural networks, or trainable quantum circuits. They all refer to the same concept [SSM21].

VQAs are strongly related to classical machine learning models. Essentially, they consist of an architecture, loss function and an optimization method. Given the drawbacks of NISQ technology, the simplicity and optimization-based approach of VQAs makes them a strong candidate for achieving a quantum advantage in the near term [CAB+21].

VQAs are trained in a hybrid setting, i.e., quantum circuits with parameters to be optimized are created on the quantum computer, whereas the optimization is done on a classical one. This mechanism has the advantage of using shallow circuits (few qubits and gates), resulting in less noise. Several other algorithms have already been developed

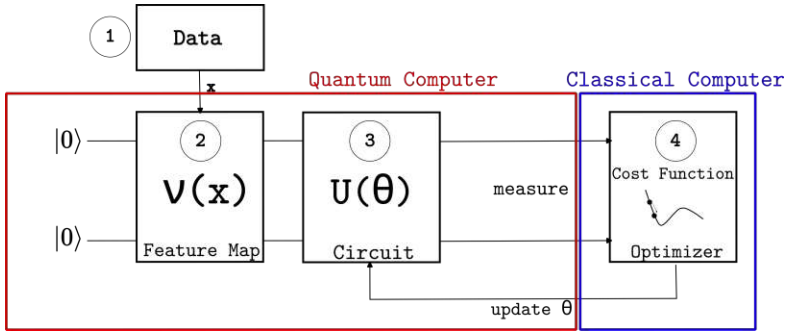


Figure 2.1: VQA: Adapted from [BBF+22]

for future fault-tolerant systems, but VQAs have the advantage of already being relevant today [CAB+21].

Essentially, they consist of (1) a cost function for a given problem, (2) an ansatz, defining the parameters to be optimized, and depending on the type of problem they can have (3) a training set. Trivially, for ML problems, a training set is required. We will dive into the individual components in the following [CAB+21].

We show an outline of a VQA in Figure 2.1. The model takes classical data as the input in step 1 and maps the data points onto qubits using a feature map in step 2. In step 3, it executes the circuit and measures the output. The result is then transferred to the classical computer in step 4, where the parameter update is determined using the cost function. The updates are then propagated to the quantum computer. Steps 3 and 4 are iteratively repeated until convergence or until a stopping criterion is fulfilled.

Cost Function The first step when building a VQA is to find a cost function. In machine learning, it is usually a straightforward process since it is standardized. However, VQAs are also relevant for other tasks, such as combinatorial optimization problems, where finding the loss function may be more challenging. The cost function is a mapping from the parameters of the circuit to a real number, where lower values indicate better solutions. The cost function is shown in Equation 2.28, where U is the circuit with parameters θ, p represents the input states from the dataset, and O represents the observables [CAB+21].

$$C(\theta) = f(p, O, U(\theta)) \tag{2.28}$$

A good cost function should meet several criteria, including being faithful (i.e., zero cost represents the optimal solution), allowing for estimation by classical post-processing of measurement outcomes, producing meaningful values (with smaller values indicating better solutions), and being amenable to optimization. Additionally, with respect to NISQ devices, the circuit depth should be kept small. The cost function is a critical

component of the algorithm, and its effectiveness determines the success of the algorithm in finding optimal solutions [CAB+21].

Ansatz The ansatz is a critical component of any VQA as it defines the architecture and parameters of the quantum circuit. The choice of ansatz is often dependent on the specific task at hand, with some ansatzes designed specifically for certain problems, while others are more general. Essentially, an ansatz can be thought of as a sequence of unitary transformations applied to the quantum state. The choice of ansatz can significantly impact the performance of the algorithm, with certain ansatzes being better suited to particular problems than others [CAB+21]. We will delve deeper into this issue in a later section.

Training Set The training set used for the VQA depends on the type of problem being addressed, whether supervised or unsupervised, or not related to machine learning at all. Essentially, it is a set of instances that serve as input to the algorithms, though they need to be encoded into a quantum state first [CAB+21].

2.3.4.1 Applications

VQAs have become popular in a variety of field, including mathematics, machine learning or optimization. In particular, researchers in [Bia21] have shown that they are universal with respect to other quantum computations, i.e., theoretically, they are equally powerful as any other quantum algorithm. In the following, we list several possible applications [CAB+21].

- *Ground and exciting states*: VQAs have so-far been used to estimate the energy of a molecule. These are encoded into a Hamiltonian, and the task at hand is to estimate the eigenvalue and eigenstates of it.
- *Optimization*: Similarly, it is possible to use VQAs to solve optimization problems. More specifically, several methods for tackling combinatorial optimization (such as the MAX-SAT problem) have been proposed. Among the most important is the quantum approximate optimization algorithm (QAOA) [FGG14].
- *Mathematics*: VQAs have also been used to solve mathematical problems, such as (non-)linear equations or integer factorization. Although there exist exact algorithms, such as Shor's algorithm [Sho97] for integer factorization, these cannot be sufficiently correctly implemented on NISQ technology.
- *Compilation*: Given that so-far shallow quantum circuits are necessary to get accurate results, a lot of thought has to go into the compilation. Given the complexity of the problem, they are difficult to optimize on classical computers.

- *Error correction*: Although error correction algorithms exist, they come with a large overhead and cannot be implemented on NISQ technology. Still, researchers have suggested using VQAs to mitigate errors on a small scale (e.g., [JRO+17]).
- *Machine learning*: Given the already mentioned similarity between classical ML models and VQAs, it is hardly a surprise that they are frequently used for machine learning as well.

2.3.4.2 Ansatzes

An ansatz serves as a blueprint for defining the parameters θ that are to be optimized. It also outlines the general architecture for solving a problem, which can be specific to a particular problem or more general. Typically, an ansatz is expressed as a unitary, where the parameters are part of the unitary itself. However, when using NISQ devices, it is crucial to consider the circuit depth and overhead required to implement the ansatz. There are different types of ansatzes that are designed for different purposes, and each of them has its strengths and weaknesses [CAB+21]. We will introduce a few in the following.

A hardware-efficient ansatz is a type of ansatz with a small depth and minimal overhead. These circuits are versatile and can be used in various applications. However, one must be cautious when initializing them, as random initialization can lead to problems. Another type of ansatz is the unitary-coupled cluster ansatz, which is particularly important in quantum chemistry [CSV+21].

The quantum alternating operator ansatz, on the other hand, is universal for sets of graphs and hyper-graphs. This ansatz is inspired by the quantum approximate optimization algorithm (QAOA) and has the advantage of having a feasible subspace that is smaller than the full Hilbert space. This feature may lead to better performance. The variational Hamiltonian ansatz is also inspired by the QAOA ansatz and is very versatile, often used in quantum chemistry, optimization, and simulation. It is designed to optimize the parameters of a Hamiltonian, and its versatility makes it suitable for various applications [CSV+21].

The variable structure ansatz changes circuit elements instead of just the parameters, making it more flexible than other ansatzes. This type of ansatz was initially introduced in adapt-VQE [GEBM19], and it presents a challenging problem for optimization due to the large search space. Finally, the sub-logical ansatz and quantum optimal control include device-level parameters in the parameters, providing additional flexibility. There is evidence that this can help mitigate the effects of noise.

In the following, we present four popular ansatzes for QML, namely, Two-Local, Real Amplitudes, EfficientSU2, and Pauli Two-Design. These ansatzes share a common structure, starting with a layer of single-qubit rotations followed by entanglement layers. The details of each ansatz can be found in the documentation, which we reference for a more comprehensive description.

Two-Local The Two-Local ansatz is a variational quantum circuit composed of repeated rotation and entangling layers. The rotations are implemented using single-qubit gates, while the entangling gates are two-qubit gates. The specific rotations and entangling gates can be customized to suit the problem at hand. Overall, the Two-Local ansatz provides a flexible tool for constructing variational circuits that can be used for a wide range of quantum applications[7]. We show the ansatz in Figure 2.2a. The $R_{X|Y|Z}$ gates represent rotations around the X, Y or Z axis of the Bloch sphere, which is a way to represent a qubit geometrically. The angle is defined by the learnable parameters θ. The gates spanning two qubits represent entanglement gates, in this case controlled-X (CNOT) gates.

Pauli Two-Design The Pauli Two-Design ansatz is a variational that is derived from the Two-Local ansatz. It involves repeated entanglement layers that use pairwise controlled-Z gates to enable efficient quantum computation. The ansatz starts with an initial rotation around the Y-axis with an angle of $\frac{\pi}{4}$. The subsequent layers consist of alternating rotations around the X, Y, or Z axis, which are chosen at random, and entangling layers[8]. We show the ansatz in Figure 2.2b. It is built similarly to the Two-Local ansatz, except that it uses controlled-Z gates, instead of controlled-X ones. Furthermore, the qubits are rotated with an angle of $\frac{\pi}{4}$ around the Y axis in the beginning.

Real Amplitudes The Real Amplitudes ansatz is derived from the Two-Local ansatz too. It creates a trial wave function by repeatedly applying Y rotations to the qubits and controlled-X gates to entangle them. As the name implies, the amplitudes of the wave function are restricted to be real, which can simplify the optimization problem in some cases[9]. The ansatz is shown in Figure 2.2c.

EfficientSU2 The EfficientSU2 ansatz is a hardware-efficient variational quantum circuit. It uses single-qubit transformations, including rotations around the X, Y, and Z axes, as well as two-qubit entangling gates like the controlled-not (CNOT) gate, to create the circuit. The name SU2 refers to the special unitary group of size 2, which includes all 2×2 complex matrices with determinant 1. The EfficientSU2 ansatz can also be repeated multiple times to create a deeper circuit and increase its expressive power[10]. We show the ansatz in Figure 2.2d. It can be seen that it uses a lot more rotation gates than the other ones do, using both Y and Z rotation gates in every repetition.

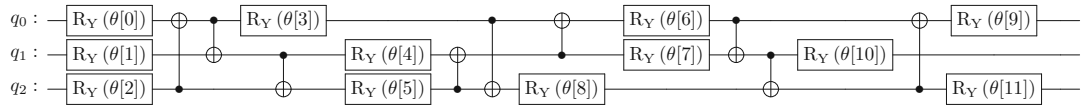
The entanglement strategy for the CNOT gates used in the Two-Local, Real Amplitudes, and EfficientSU2 ansatzes can be specified. Such include full entanglement, linear

[7] https://qiskit.org/documentation/stubs/qiskit.circuit.library.TwoLocal.html#qiskit.circuit.library.TwoLocal. Accessed 02.05.2023

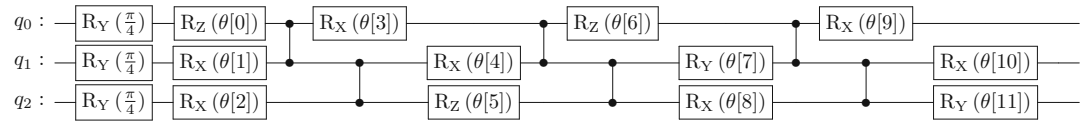
[8] https://qiskit.org/documentation/stubs/qiskit.circuit.library.PauliTwoDesign.html#qiskit.circuit.library.PauliTwoDesign. Accessed 02.05.2023

[9] https://qiskit.org/documentation/stubs/qiskit.circuit.library.RealAmplitudes.html#qiskit.circuit.library.RealAmplitudes. Accessed 02.05.2023

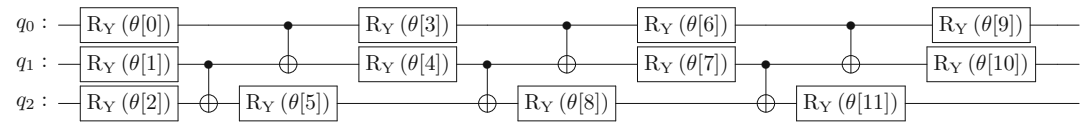
[10] https://qiskit.org/documentation/stubs/qiskit.circuit.library.EfficientSU2.html#qiskit.circuit.library.EfficientSU2. Accessed 02.05.2023



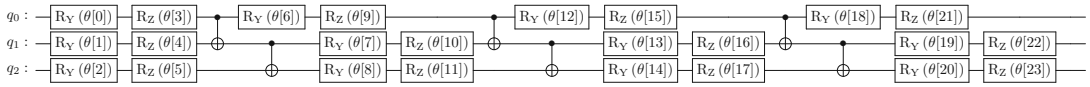
(a) TwoLocal



(b) PauliTwoDesign



(c) RealAmplitudes



(d) EfficientSU2

Figure 2.2: Ansatzes

entanglement (where each qubit is entangled with its subsequent neighbor), reverse-linear entanglement, pairwise entanglement (where qubits are paired, and each pair is entangled in alternating layers), circular entanglement (which is similar to linear entanglement, but the last qubit is also entangled with the first qubit) and sca entanglement (shifted-circular-alternating, which is essentially circular, but the connection between the last and first qubit is shifted by one and the control and target qubits are swapped in each block)[7].

2.3.4.3 Optimization

Similarly to the optimization of classical NNs, the optimization of VQAs on classical computers is a non-convex problem (NP-hard), meaning it has many local optima, making it computationally expensive. Additional to the problems faced in classical NNs, there are other challenges such as noise, and barren plateaus (BPs) (introduced later on) that make optimization even more difficult [CAB+21].

One approach to optimize VQAs is to use gradient-based methods. These methods involve taking a step in the opposite direction of the gradient, which indicates the direction of steepest descent. One such method is Stochastic Gradient Descent (SGD),

which is inherited from classical ML and backpropagation [RHW86]. Another popular method is Adam [KB15], which adapts the step size to achieve faster convergence. Furthermore, natural gradient descent [Ama98] is a gradient-based method that works on a space that takes into consideration the changes in the loss function when changing the parameters [CAB+21].

Another approach to optimization is through meta-learning, i.e. learning to learn. Meta-learning involves training a NN to make good updates based on past and gradient information. This approach has high potential and can be useful for optimizing VQAs. Another frequently used method is Simultaneous Perturbation Stochastic Approximation (SPSA). This method takes one partial derivative in a randomly chosen direction, which makes it more efficient than computing the gradient [CAB+21].

There are various optimization approaches available for VQAs, including gradient-based methods, meta-learning, and gradient-free methods, each with their own strengths and weaknesses. Overall, optimizing VQAs on classical computers is a challenging task due to the non-convexity of the problem, noise, and BPs. However, due to the many local extrema in the optimization landscape, convergence guarantees are difficult to achieve [CAB+21].

The three optimizers we will consider are COBYLA, SPSA, and Nelder-Mead, which will be introduced in the following. Each of these optimizers has different characteristics that make them suitable for different types of optimization problems. They are widely-used for QML and we believe that their different characteristics make them interesting candidates for our experiments.

COBYLA [Pow94] is a gradient-free optimization algorithm commonly used in QML due to its ability to handle numerous constraints. Nelder-Mead [NM65] is a simplex-based optimization algorithm that is well-suited for low-dimensional optimization problems and is commonly used in QML for tasks such as variational quantum eigensolver (VQE) calculations. SPSA [Spa98] is a gradient-based method that often works well in presence of noise. In fact, in the Qiskit documentation, it is recommended to use SPSA when working with noise[11].

COBYLA The Constraint Optimization BY Linear Approximation (COBYLA) algorithm was first introduced in [Pow94]. It is an iterative, gradient-free optimization algorithm. In each iteration, COBYLA estimates the objective function and creates a trust region surrounding the current solution candidate. Within this region, the algorithm constructs an approximation of the objective function based on function evaluations. The trust region size is then adapted based on the agreement between the approximation and the true function values, which determines the allowable range of movement for the current solution candidate in the subsequent iteration. The solution candidate is then

[11] `https://qiskit.org/documentation/stubs/qiskit.algorithms.optimizers.SPSA.html`. Accessed 17.06.2023

moved to the minimum within the trust region. By iteratively refining the approximation and adjusting the trust region, COBYLA converges towards the optimal solution.

SPSA The simultaneous perturbation stochastic approximation (SPSA) [Spa98] algorithm is a gradient-based method. It offers the advantage of calculating the objective function using only two measurements, irrespective of the number of parameters involved in the optimization problem. This property makes the SPSA algorithm an efficient optimization technique. The author suggested that when dealing with many parameters, the SPSA algorithm may serve as a favorable alternative due to its ability to provide effective optimization results with minimal measurements.

Nelder-Mead The Nelder-Mead algorithm [NM65] is a popular gradient-free optimization method. It works by maintaining a simplex, which is a set of $n + 1$ points in the search space that form a geometric shape resembling a simplex (a triangle in 2D, a tetrahedron in 3D, etc.). The vertex with the highest objective is replaced by another vertex, leading to the exploration of the search space. The algorithm terminates when the simplex becomes sufficiently small, meaning that the points are close together.

The optimization of VQAs faces a significant obstacle known as the barren plateau (BP). This issue causes the loss landscape to become flat, which renders gradient-based and gradient-free methods ineffective since there is no information on where to go. The problem is particularly pronounced when using a general ansatz, regardless of the problem's complexity, due to the vast search space. In most cases, initializing randomly makes it almost impossible to find the optimal solution [MBS+18].

Furthermore, the barren plateau (BP) problem is known to be dependent on the cost function used. Global cost functions suffer more from this problem than local cost functions [CSV+21]. Noise can also contribute to the formation of BP, further aggravating the problem [WFC+21]. To overcome this issue, careful selection of ansatz architectures that minimize the occurrence of the BP is necessary [CAB+21].

2.3.4.4 Challenges and Opportunities

One of the major challenges in implementing VQAs is trainability, which is often hindered by the occurrence of the BP. A lot of research is currently going into methods that try to mitigate or overcome this plateau [CAB+21].

Another important challenge in developing VQAs is ensuring efficiency. This means that the cost function and the expectation values need to be efficiently calculable. In addition, the occurrence of BPs may require higher precision, particularly in measurements, which are crucial for the success of the algorithm. One strategy to address this is to reduce the number of measurements needed, for example, by partitioning into simultaneously measurable subsets. We will not go into detail of such strategies, the interested reader may refer to [CAB+21] for further information.

Another critical challenge in VQAs is ensuring accuracy, especially for near-term quantum devices like NISQ computers that have high levels of noise and errors. To address this, error correction and mitigation techniques are essential. Noise can affect the optimization landscape and make the optimal solution harder to find. The presence of noise can slow down the training process by reducing the gradients, which becomes more severe with larger circuits. To overcome these effects, error mitigation techniques can be applied to correct for errors and improve the accuracy of the algorithm. However, it is worth noting that VQAs are inherently noise-resilient, and there are methods available, such as zero-noise extrapolation, which allow for the estimation of error-free results based on measurements at different noise levels [CAB+21].

VQAs offer exciting opportunities for solving complex problems in various fields, including chemistry, nuclear, and particle physics [CSV+21]. In chemistry, VQAs can be used to simulate molecules and electronic systems, which is particularly relevant to protein folding and drug-receptor interactions [CRAG18, CRO+19, OSS+21]. In nuclear and particle physics, VQAs have the potential to solve a range of problems, including finding nuclear ground states [DMH+18, LKL+19] or simulating neutrino-nucleon scattering [RLC+20].

The large dimension of the Hilbert space in quantum computing provides a unique opportunity for both optimization and machine learning tasks, due to being able to store a lot of data. One leading candidate for combinatorial optimization is the Quantum Approximate Optimization Algorithm (QAOA) [FGG14]. Quantum neural networks can have a higher capacity than comparable classical NNs, as measured by the effective dimension [ASZ+21]. Additionally, even deep reinforcement learning has been proposed as a way to utilize the power of quantum computing in training complex models [CYQ+20].

QML is an exciting and rapidly growing field with many challenges and opportunities for innovation. As quantum computing technology continues to advance, the potential for groundbreaking applications in areas such as drug discovery, financial modeling, and data analysis is becoming increasingly evident. With its potential to solve previously intractable problems, quantum machine learning is a field with immense promise [Pas23].

CHAPTER 3

Related Work

In the following chapter, we will delve into related work on quantum machine learning (QML). Specifically, we will discuss research related to three main aspects: feature encoding, optimization, and variational quantum algorithms (VQAs). First, we will examine recent work for encoding classical data into quantum states, which is a crucial step in quantum machine learning, in Section 3.1.

Next, we will take a look at recent progress and important research areas with respect to optimization in Section 3.2 and the ansatz in Section 3.3. Finally, we will explore state-of-the-art research on VQAs in general in Section 3.4.

Through this comprehensive review of related work, we aim to provide a clear understanding of the current state-of-the-art in quantum machine learning and the challenges that lie ahead.

In general, according to Alchieri et al. [ABBB21], one can distinguish two different approaches in QML, namely (1), the *translational approach*, where classical machine learning models are converted to quantum ones with the goal to obtain a speed-up and (2), the *exploratory approach*, which has the goal to exploit the underlying quantum mechanics to train a model. The latter are usually more innovative, as they are specially built to exploit quantum effects.

Another way of categorizing current research is by the benefit researchers are looking for. These include (1) computational complexity, (2) sample complexity (number of data points to get a model that generalizes), (3) robustness to noise (in the data, not the computer) and (4) model complexity [ABBB21].

In [OM20], the authors provide an extensive review about recent advances in QML. This includes notable advancements such as the implementation of a neural network based on the perceptron algorithm as described in the work by Tacchino et al. [TMGB19]. Another significant development highlighted in the review is the implementation of a quantum

support vector machine, which achieved an exponential speed-up compared to classical counterparts, as demonstrated by Rebentrost et al. [RML14]. For further information on the development of QML, we refer to the aforementioned review. In our discussion here, we will mainly discuss work related to VQAs.

3.1 Feature Encoding

Numerous research initiatives have delved into exploring quantum support vector machines and feature maps due to the similarity of QC and kernel methods. Suzuki et al. [SYG+20] provide a lower bound on the training accuracy of a kernel-based quantum classifier for two qubits. Notably, they only use the input dataset and the feature map to obtain the lower bound. However, the authors question the practical utility of the lower bound, due to computational scaling. The calculations scale with the dimension of the Hilbert space, which is exponential in the number of qubits. Nevertheless, despite the computational hurdles, the ability to calculate reasonable lower bounds on the training accuracy is intriguing and contributes to the understanding of the potential capabilities of kernel-based quantum classifiers.

Daspal provides a method called 'OptiPauli' to find the optimal Pauli feature map in [Das22]. The Pauli feature map involves applying sequences of Pauli transformations, such as X, Y, and Z rotations, to the input data. The optimization of the feature map can become computationally challenging due to the large search space. Therefore, the authors split the problem into subproblems and employ a genetic algorithm to optimize the feature map.

In the context of feature encoding, various theoretical proofs have been established. Knowing that classical feed-forward neural networks with just one hidden layer are universal approximators [HSW89] (which has been extended to many more architectures), it is interesting to find out in how far this applies to VQAs. In the paper by Goto et al. [GTN21], the authors show that quantum feature maps, in their typical configuration, are universal approximators of continuous functions.

Furthermore, Schuld et al. [SSM21] investigate the expressiveness of a VQA. Specifically, they explore the representation capabilities of VQAs with a single variable. The authors demonstrate that such VQAs can be represented as partial Fourier series, where the feature encoding determines the accessible frequencies, and the ansatz influences the coefficients. In their analysis, the authors highlight that a single qubit rotation limits the frequency to only one sine wave. Then, they continue to show that VQAs can, given a sufficiently wide frequency spectrum, act as universal function approximators. They conclude that the number of rotations applied to the circuit has a great influence on the expressiveness of the model. The models tend to learn periodic functions, and appropriately scaling the data and preprocessing techniques can impact the accessible frequency range [SSM21].

Casas and Cervera-Lierta [CCL23] extend this approach to VQAs with multiple qubits

(meaning multi-dimensional data). They find that for multi-dimensional data, the degrees of freedom of the Fourier series surpass the tunable parameters in the model. Therefore, it cannot be generalized that VQAs can approximate any multi-dimensional Fourier series.

Metric learning takes a distance metric, i.e. the L2 distance, and tries to maximize the distance between data points of different classes. Xing et al. [XNJR02] established the foundation of this field by demonstrating that metric learning can be formulated as a convex optimization problem. Lloyd et al. [LSI+20] propose a quantum version, *quantum metric learning*, that tries to maximize the distance in the Hilbert space. The authors argue that when suitable feature maps can be constructed, the corresponding ansatz can be relatively simple and implemented using shallow circuits. The results are difficult to reproduce classically, but the authors recognize that they are not sure whether it will give a quantum advantage.

To reduce the number of qubits required and the feature encoding time, researchers continuously explore new techniques. Ovalle-Magallanes et al. [OMACAC+23] propose a novel feature map for quantum convolutional neural networks (which employ VQAs), which reduces the number of qubits from $O(N)$ to $O(\lceil log_2(N) \rceil), N = k \times k$ (k being the filter size).

Sierra-Sosa et al. [SSPT23] conduct a study investigating the influence of data, classical transformations on data, and feature encoding techniques on QML models. In their research, they focus on two specific feature encoding methods, namely, amplitude encoding and the previously mentioned feature map based encoding. To evaluate the impact of different rotations on the models, the researchers generate datasets and apply various rotations to them. They observe that amplitude encoding is less affected by the different rotations compared to the feature map based one. Interestingly, they find that rotating the dataset can actually improve the results. Based on their findings, the authors propose a strategy of rotating the dataset to create a different representation. The approach aligns with the idea presented by Schuld et al. [SSM21], suggesting that appropriately scaling the data may be beneficial in QML [SSPT23].

Furthermore, in the field of QML, feature reduction is frequently employed due to the limited number of available qubits. Mancilla and Pere [MP22] employ classical and quantum ML and compare different feature reduction techniques for the quantum models. In their study, the authors find that linear discriminant analysis (LDA) outperforms principal component analysis (PCA) on their two datasets when applied in the context of quantum machine learning. Hancco-Quispe et al. [HQBCTC22] report the same results. However, at present, there are no hypotheses or proofs provided, explaining the superior performance of LDA compared to PCA in these scenarios.

3.2 Optimizer

Optimization plays a crucial role in VQAs, highlighting the significance of this area of study. The barren plateau (BP) stands out as a common occurrence during optimization processes, where the cost function fails to provide information about the location of the minimum even when the current candidate if far away from it. Consequently, extensive research is dedicated to exploring optimization methods, such as the utilization of Bayesian learning for improved parameter initialization by Rad et al. [RSL22]. Moreover, novel optimization techniques are frequently reported in the literature. For instance, Boyd et al. [BK22] introduce a method that calculates covariances and estimating derivatives at each step.

Schuld et al. [SBG+19] examine the significance of gradients in optimization processes and propose a technique for estimating gradients of expectation values using a similar or nearly identical VQA. The study suggests that it is often sufficient to run the original circuit twice, with a single gate parameter shifted, to obtain the component in the gradient.

Moreover, a study by Bittel and Kliesch [BK21] on the asymptotic time complexity of optimizing VQAs demonstrates that the classical optimization problem is NP-hard. It is robust, meaning that for every polynomial algorithm, there are instances where the relative error can be arbitrarily large (assuming $P \neq NP$). The authors additionally emphasize the presence of numerous local optima in the loss function of VQAs, causing algorithms to frequently converge far from the optimal solution. The convergence point heavily depends on the initialization.

Joshi et al. [JKA21b] investigate local optimizers in VQAs using EfficientSU2 as the ansatz and ZFeatureMap as the feature map. The researchers compare the performance of VQC with AQGD (Analytic Quantum Gradient Descent, a variant of classical gradient descent) and COBYLA to classical machine learning models, namely SVM, GB, and RF, in analyzing sentiment data. The study finds that the VQA with AQGD outperforms the other models on this task.

In another study by Bonet-Monroig et al. [BMWV+23], the authors compare the performance of four optimization algorithms (SLSQP, COBYLA, CMA-ES, and SPSA, for information please refer to the paper) in finding ground state energies in chemistry and material science tasks. The study highlights the importance of hyperparameter tuning of optimizers in VQCs. They find that CMA-ES can be tuned to outperform SPSA. The need for tuning increases with increasing noise in the circuit, and the landscape is highly dependent on the setup and problem. Furthermore, the study reveals substantial differences in performance through hyperparameter tuning for CMA-ES, whereas SPSA demonstrates relatively little sensitivity, suggesting that SPSA may possess a more versatile nature and applicability across different scenarios.

Huembeli and Dauphin [HD21] discuss the open questions regarding convergence and trainability of VQAs and introduce a technique that computes the Hessian. The primary

objective is to identify effective approaches for analyzing the loss landscape in these models. They argue that analyzing the Hessian and its eigenvalues and eigenvectors can lead to a better understanding of the landscape and the optimization process as a whole. However, a significant limitation lies in the computational expenses associated with implementing this technique.

Another study by Sung et al [SYH+20] proposes two novel surrogate model-based optimization methods, namely Model Gradient Descent (MGD) and Model Policy Gradient (MPG). Both MGD and MPG are iterative algorithms that involve sampling from the environment surrounding the current candidate solution and constructing quadratic models. These quadratic models are then utilized in the optimization process. The study reveals that stochastic optimizers such as MGD, MPG, and SPSA exhibit greater resilience to variations in problem domains. On the other hand, deterministic optimizers can benefit from tuning but are more reliant on specific variations. The authors emphasize the importance of relevant cost models and hyperparameter optimization of optimizers.

Rivera-Dean et al. [RDHAB21] present a unique approach, using a classical neural network to help optimize a VQA. Interestingly, the NN utilized in this context is unrelated to the specific problem being addressed. Instead, its purpose is to modify the landscape of the loss function, thereby preventing the optimizer from becoming trapped in local optima. The authors apply this method to solve a Max-Cut problem and observe that their approach converges in deeper local optima compared to the original algorithm.

3.3 Ansatz

Moreover, a significant amount of research is focussed on the optimization of the ansatz. Du et al. [DHY+22] argue that, while more complex ansatzes lead to better results, the noise introduced by the many gates compensates for the improved results. Consequently, they propose a quantum architecture search method, inspired by classical neural architecture search, for VQAs. This approach aims to design an ansatz tailored to a specific problem, taking into account the noise inherent in quantum computers. Similarly, Bilkis et al. [BCV+23] propose a variable ansatz, that dynamically adds or removes gates based on predefined rules. Their objective is to optimize the trainability of the ansatz and address challenges related to noise in quantum systems.

In 2019, Grimsley et al. [GEBM19] introduced the ADAPT-VQE algorithm for building ansatzes for the simulation of chemical systems. They start with a very shallow ansatz and adaptively add operators, depending on the molecule being simulated. Patil et al. [PWK22] build on top of ADAPT-VQE and introduce an adaptive ansatz, focussing on solving a system of linear algebraic equations. Their objective is to adapt the architecture step by step until a certain performance is achieved. Concurrently, they strive to minimize the computational resources required for the task.

Ostaszewski et al. [OTM+21] use reinforcement learning to optimize the ansatz of a variational quantum eigensolver. Their approach focuses on adapting the complexity of

the ansatz based on the specific problem at hand, while aiming to maintain a low circuit depth to minimize the effect of noise. The proposed method is applied to estimate the ground-state energy of lithium hydride, resulting in state-of-the-art outcomes. Moreover, Zhao et al. [ZTK+20] address the vast number of measurements that need to be done in a VQA. They use a unitary partitioning [IYLV20] approach to reduce the number of measurements.

Furthermore, Choquette et al. [CDPB+21] introduce a novel class of ansatzes, called quantum-optimal-control inspired ansatzes. They demonstrate that, on certain problems, these ansatzes exhibit faster convergence compared to traditional ansatzes. While common ansatzes aim to preserve the symmetry of the Hamiltonian to limit the search space, the authors argue that maintaining symmetry does not always provide an advantage. Consequently, their approach intentionally seeks to break the symmetries in order to explore a broader range of possibilities. They showcase the ansatz for physics and chemistry problems and show that it performs well in these settings.

Du et al. [DTYT22] measure the expressivity of a VQA using the concept of a covering number. They state that the expressivity of a VQA is heavily influenced by the choice of gates and measurements employed. Notably, when utilizing NISQ devices, they find that the expressivity of the VQA diminishes as the circuit depth increases. Using their measurement for expressivity in simulations, they conclude that difficulties arise with insufficient or excessive expressivity.

3.4 Variational Quantum Algorithms

Finally, we study literature on parameter configurations on VQAs, similarly to the experiments we conduct. Suryotrisongko and Musashi [SM22] use a security dataset to compare the performances of a novel hybrid deep learning model and a VQA. They consider a variety of different feature map, ansatz and optimizer combinations for the VQA and find significant differences in performance. Furthermore, their novel architecture outperforms the VQA on this dataset.

Piatrenka and Rusek [PR22] perform experiments using different configurations for the iris dataset. They experiment using ZFeatureMap, ZZFeaturemap and a PauliFeaturemap with gates ['Z', 'Y', 'ZZ']. Regarding the ansatz, they only experiment using the RealAmplitudes one, but try different numbers of repetitions. For the optimizer, they consider SPSA, COBYLA and Sequential Least SQuares Programming (SLSQP). They obtain the best results using one repetition of the PauliFeatureMap and two repetitions of the RealAmplitudes ansatz. Furthermore, they report that COBYLA and SLSQP outperform SPSA. The authors run their experiments on both simulators and a real quantum computer and find big performance differences.

Moreover, Katyayan and Joshi [KJ23] propose a model for classifying questions from two categories of the SelQA dataset. They use a TwoLocal ansatz using 'ry' and 'rz' as the rotation gates, and 'cz' as the entanglement ones. Furthermore, they experiment with all

three of Qiskit's feature maps beforehand, finding that the PauliFeatureMap works best, which they use for the experiments. They find that a small depth is advantageous and that the features strongly influence the output.

Finally, Joshi et al. [JKA21a] compare classical ML models to quantum ones on sentiment analysis. They find that quantum models slightly outperform the classical models in their configuration. They employ a PauliFeatureMap using ['X', 'XX', 'YY', 'XY', ZXZ'] as the Pauli gates. Furthermore, they use the Efficient SU2 and RealAmplitudes ansatz, using 100 and 150 epochs. COBYLA is used as the optimizer. All in all, the EfficientSU2 ansatz with 100 epochs outperforms all other models.

CHAPTER 4

Methodology

In this chapter, we present our experimental methodology. First, we describe the datasets used for our experiments in Section 4.1. Specifically, we provide an overview of the datasets we selected, which include two regression and two classification problems.

In Section 4.2, we introduce our classical machine learning (ML) baseline. The methodology is based on traditional algorithms and techniques that are commonly used in the field of ML. Such include grid search and cross validation.

In Section 4.3, we describe our proposed quantum machine learning (QML) methodology. We provide a detailed explanation of our QML approach. In particular, we employ an exhaustive search on a subset of the training data to tune the most important hyperparameters.

4.1 Data

We choose the following datasets to ensure their suitability for our research objectives. We select low-dimensional datasets to account for the limited public availability of current quantum computers, allowing only seven qubits for free on IBM machines[1].

This approach allows us to obtain meaningful results, while avoiding the pitfalls associated with processing high-dimensional data. If we had chosen data with higher dimensionality, we would have had to reduce much more features, which would have made the comparison with classical ML even more difficult, as the quantum models would have been trained with significantly less information.

[1]According to the 'Open Plan' at https://www.ibm.com/quantum/access-plans. Accessed: 06.04.2023

Attribute	Description	Datatype
Longitude	how far east/west a block is	real
Latitude	how far north/south a block is	real
Housing median age	median age of houses in a block in years	integer
Average rooms	avg number of rooms per household in a block	real
Average bedrooms	avg number of bedrooms per household in a block	real
Population	number of people living in the block	integer
Average occupancy	avg number of household members	real
Median income	median income of households in a block	real
Median house value	in hundreds of thousands of dollars	real

Table 4.1: California Housing Dataset

4.1.1 California Housing

The California Housing dataset[2] [KPB97] is a well-known public dataset commonly used for regression tasks. The dataset contains features that describe block groups (geographical units for which the U.S. Census Bureau publishes data) with between 600 and 3,000 people in California at around 1990. The dataset has 8 numeric features, a real-valued target and 20,640 instances. The goal is to predict the median house value in \$100,000. The dataset can be loaded through the Scikit-learn [PVG+11] function `fetch_california_housing()`.

In particular, Table 4.1 shows all features provided. We list the names with a short description and datatype. The dataset description can be found in the Scikit-learn user guide[3].

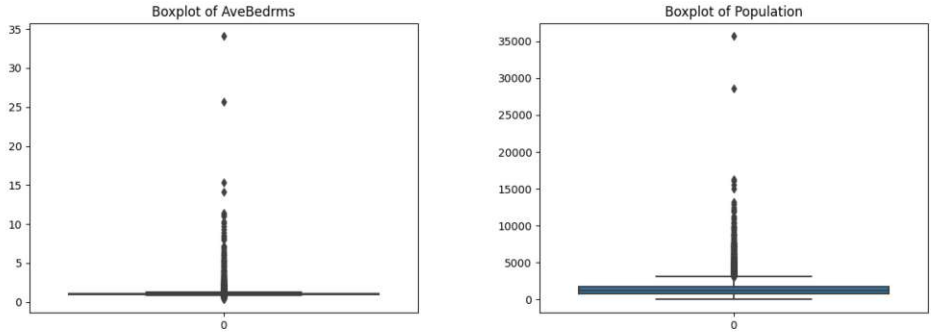
There are no missing data points in the dataset. Furthermore, we examine summary statistics of the different attributes. Interestingly, the standard deviation is very high, and the associated minimum and maximum values are far away from the mean for population and average occupancy. We give an overview of the summary statistics in Appendix B in Table B.1.

Furthermore, when looking at box plots of the different columns, several outliers can be observed. This is in particular true for the average rooms, average bedrooms, population, and average occupation attributes. Examples are shown in Figure 4.1.

Moreover, pairwise correlations are shown in Figure 4.2. Although there is a high positive correlation between the average number of rooms and the average number of bedrooms (85%), we believe that this correlation is still small enough to not significantly affect the performance of the models negatively. Furthermore, there is a correlation of 69% between the target variable price and the median income, which seems plausible, as people with higher incomes can afford more expensive houses. Lastly, the high negative

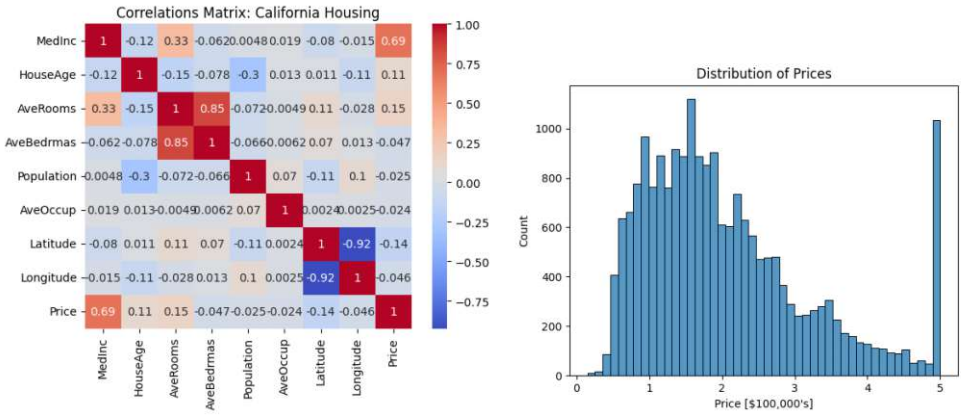
[2] https://www.dcc.fc.up.pt/ ltorgo/Regression/cal_housing.html. Accessed: 21.03.2023

[3] https://scikit-learn.org/stable/datasets/real_world.html#california-housing-dataset. Accessed: 22.03.2023



(a) Average Bedrooms (b) Population

Figure 4.1: California Housing: Distribution of Bedrooms and Population



(a) Correlations (b) Distribution of Target

Figure 4.2: California Housing Dataset Characteristics

correlation of 92% between latitude and longitude can be explained by the geographical shape of California. Blocks in the south (smaller latitude) are further in the east (higher longitude) and vice versa.

Moreover, we take a look at the distribution of the target variable in Figure 4.2b. We can see that there is a peak at $\$100,000 - \$200,000$. There are significantly fewer houses with a price between $\$300,000 - \$480,000$, but many are in the last bucket. We hypothesize that the models could have difficulties predicting the price of these houses and bad prediction could, when taking the mean squared error as the measurement, significantly impact the results.

After loading the data, we randomly split it into a training, validation and test set of size 60%, 20%, and 20% respectively and persist it to ensure we use the same splits for all experiments.

Attribute	Description	Datatype
Date		string
Hour	of the day	integer
Rented Bike Count	target	integer
Temperature	in Celsius	real
Humidity	in %	integer
Wind Speed	in m/s	real
Visibility	in 10m	integer
Dew Point Temperature	in Celsius	real
Solar Radiation	in MJ/m^2	real
Rainfall	in mm	real
Snowfall	in cm	real
Seasons	current season	categorical
Holiday	current day is (not) a holiday	categorical
Functional Day	functional vs. non-functional hours	categorical

Table 4.2: Seoul Bike Sharing Dataset

4.1.2 Seoul Bike Sharing

The Seoul Bike Sharing dataset [SY20, SJY20] is available at the UCI Machine Learning Repository[4] [DG17]. The dataset consists of 13 attributes, an integer target and 8,760 instances. The goal is to predict bike sharing demand for every hour of the day based on the weather conditions.

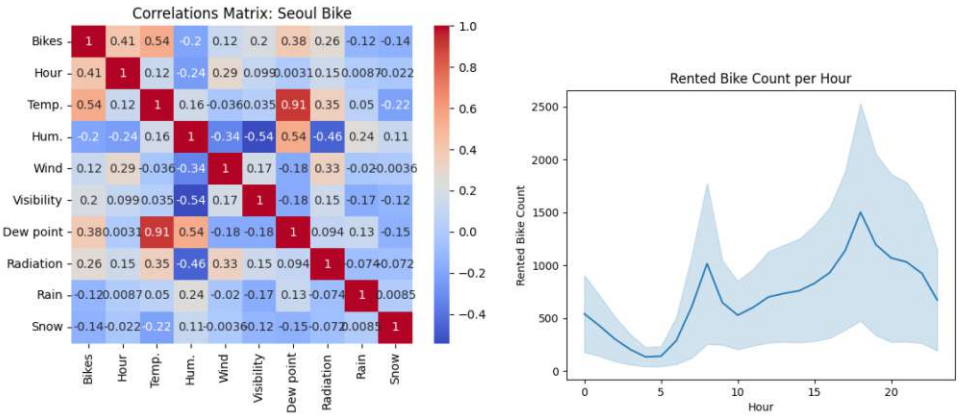
The dataset description can be found at the UCI Machine Learning Repository[4]. Table 4.2 gives an overview of the attributes, short descriptions and data types.

There are no missing data points. Table B.2 in Appendix B shows the summary statistics of the dataset. We can see that the target variable has a big span (between 0 and 3,556, with a mean of 704), which might make the prediction difficult. The temperature, dew point temperature, humidity, and visibility attributes have a high standard deviation.

Figure 4.3a shows the correlation matrix for the attributes. We can see that the dew point temperature and temperature are highly correlated (91%). It might be favorable to remove one of the attributes in the experiments to prevent the collinearity from affecting model performance. Furthermore, the rented bikes have a correlation of 54% with the temperature, which seems plausible, as does the dew point with the humidity (54%). The highest negative correlation is between visibility and humidity (-54%).

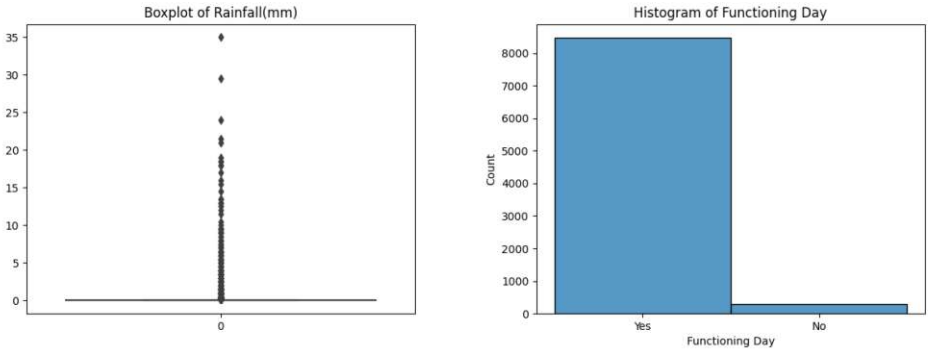
The distribution of the target variable is very unequal for this dataset, as there are a lot of very small values, especially in the morning hours. Therefore, we plot instead the mean and standard deviation of bike count per hour, which can be seen in Figure 4.3b. It can be seen that the standard deviation is smaller in the morning hours (2-6), but

[4]https://archive.ics.uci.edu/ml/datasets/Seoul+Bike+Sharing+Demand. Accessed: 21.03.2023



(a) Correlations (b) Bike Count vs. Hour

Figure 4.3: Seoul Bike Sharing Dataset Characteristics



(a) Rainfall (b) Functioning Day

Figure 4.4: Seoul Bike Sharing: Rainfall and Functioning Day

the target shows a high standard deviation in the afternoon and evening. Therefore, we believe that the hour attribute will be very important to the models.

To check for outliers, we examine the box plots of the numerical variables. We find that the rainfall and snowfall attributes have particularly many outliers. Furthermore, we look at the distribution of the categorical variables. The distribution of holiday and functioning day are very unequal. Figure 4.4 shows the box plot for rainfall and a histogram for functioning day.

After loading the data, we split the data into a training (60%), validation (20%), and test (20%) set, keeping in mind that one day has multiple entries, one for each hour. To avoid data leakage, we use a group split to ensure all data points of the same day are in the same split. Afterward, we store the splits.

Attribute	Description	Datatype
Elevation	in meters	integer
Aspect	in degrees azimuth	integer
Slope	in degrees	integer
Horiz. hydrology distance	nearest surface water features in meters	integer
Vert. hydrology distance	nearest surface water features in meters	integer
Horiz. roadway distance	distance to nearest roadway in meters	integer
Hillshade 9am/noon/3pm	index at time in summer solstice	integer
Horizontal fire point distance	nearest wildfire ignition points in meters	integer
Wilderness area	wilderness area designation (4 types)	categorical
Soil type	soil type designation (40 types)	categorical
Cover type	cover type designation (1-7)	categorical

Table 4.3: Cover Type Dataset

4.1.3 Cover Type

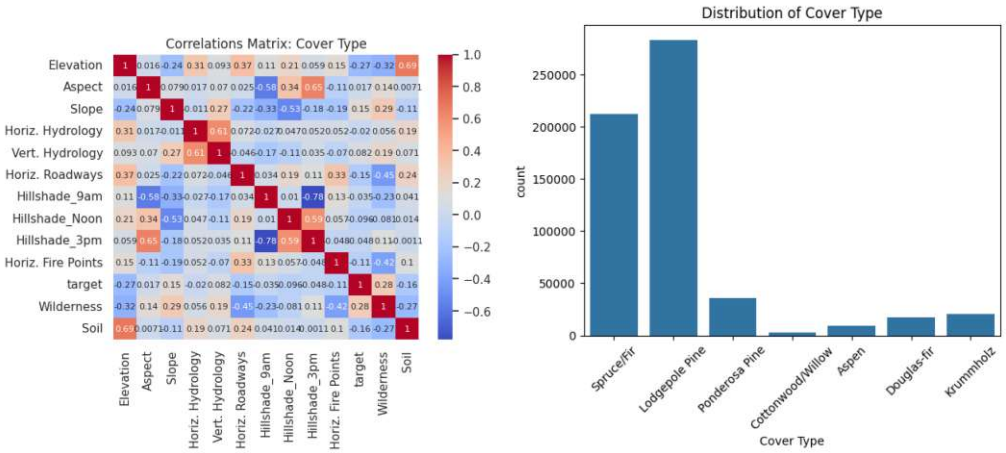
The cover type dataset[5]can be found in the UCI Machine Learning Repository [DG17]. It is a well-known classification dataset, where the goal is to predict the cover type of 30 times 30 meter forest squares based solely on cartographic attributes. It can be loaded through the Scikit-learn function `fetch_covtype()` and comprises 581,012 data points.

Table 4.3 summarizes the different attributes, descriptions and datatype. The information can be found on the UCI Machine Learning Repository page[5] and the linked dataset description.

The following represent the different classes to be predicted.

1. Spruce/Fir
2. Lodgepole Pine
3. Ponderosa Pine
4. Cottonwood/Willow
5. Aspen
6. Douglas-fir
7. Krummholz

[5]From UCI Machine Learning Repository, by Jock A. Blackard, Dr. Denis J. Dean and Dr. Charles W. Anderson, 1998, https://archive.ics.uci.edu/ml/datasets/Covertype. Copyright by Jock A. Blackard and Colorado State University. Accessed: 21.03.2023



(a) Correlations (b) Distribution of Target

Figure 4.5: Cover Type Dataset Characteristics

There are no missing values. Table B.3 in Appendix B shows summary statistics. The aspect column has a high standard deviation, as do all horizontal or vertical distances. Furthermore, when comparing the minimum, maximum and mean values, one can, especially for the hill shade attributes, see that the minimum values are far away from the mean values, meaning there could be several outliers. The same applies to the slope parameter, where the mean value is at 14, standard deviation 7 and the maximum value is at 66.

Figure 4.5 shows the correlation matrix for the dataset. The biggest negative correlation is between hill shade at 9am and 3pm (-78%), which makes sense given the sun's trajectory. It is still negligible because of the relatively small value. There are no other significant correlations.

Furthermore, in Figure 4.5b, the distribution of the target variable is shown. It can be seen that it is again very unequal, which might make the prediction for the underrepresented classes difficult. Looking at the distribution of the attributes shows that the hill shade attributes do have outliers, especially on the lower spectrum. The same holds true for the slope parameter.

4.1.4 KDD Cup 1999

The KDD Cup dataset [LHF+00] is a well-known public classification dataset for intrusion detection. It consists of a set of network traffic data that was collected in a simulated environment. It has 41 features and a categorical target with 21 classes. The sample size is 494,021 and the dataset can be loaded using the Scikit-learn function `fetch_kddcup99()`.

In Table 4.4, the attributes are explained. These and the following information can be

Attribute	Description	Datatype
Duration	duration of the connection in seconds	integer
Protocol	type of protocol	categorical
Service	network service	categorical
Flag	status of connection	categorical
Source Bytes	bytes transferred to destination	integer
Destination Bytes	bytes transferred to source	integer
Land	if connection is from/to the same host/port	binary
Wrong Fragment	number of wrong fragments	integer
Urgent	number of urgent packets	integer
Count	connections to same host in past 2 seconds	integer
SYN Error Rate	rate of SYN errors of count	real
REJ Error Rate	rate of REJ errors of count	real
Same Service Rate	rate of conn. to same service of count	real
Diff. Service Rate	rate of conn. to diff. services of count	real
Serv. Count	connections to same service in 2 two seconds	integer
Serv. SYN Error Rate	rate of SYN errors of serv. count	real
Serv. REJ Error Rate	rate of REJ errors of serv. count	real
Serv. Diff Host Rate	rate of conn. to diff. hosts of serv. count	real
Hot	number of 'hot' indicators	integer
# Failed Logins	login attempts	integer
Logged In	if user logged in successfully	binary
# Comprised	comprised conditions	integer
Root Shell	if root is obtained	binary
Su Attempt	if command attempted	binary
# Root	number of root accesses	integer
# File Creations	number of file creations	integer
# Shell	number of prompts	integer
# Access Files	operations on access control files	integer
# Outbound Commands	in an ftp session	integer
Is Hot Login	if login on the 'hot' list	binary
Is Guest Login	if login is guest	binary
Attack	type of attack	categorical

Table 4.4: KDD Cup 1999 Dataset

found in task description of the KDD Cup 1999[6].

The following list the different categories. In particular, they can be divided into 4 bigger categories, Denial-of-Service, Probing, User-to-Root and Root-to-Local attacks. We will not explain the different types of attacks. Instead, the interested reader is referred to introductory cybersecurity literature. Additionally, `normal`, meaning no attack, can be

[6]https://www.kdd.org/kdd-cup/view/kdd-cup-1999/Tasks. Accessed: 25.03.2023

predicted.

- *Denial-of-Service:* back, land, smurf, teardrop, neptune
- *Probing:* ipsweep, nmap, portsweep, satan
- *User-to-Root:* buffer overflow, loadmodule, perl, rootkit
- *Root-to-Local:* ftp write, guess password, imap, multihop, phf, spy, warezclient, warezmaster

There are again no missing values. Tables B.4, B.5, B.6 and B.7 in Appendix B show summary statistics of the dataset. The distribution of the duration variable is quite interesting. The standard deviation is very high (707.75), even though the 25th, 50th and 75th quantiles are all at 0. The maximum value lies at $58,329$. The count and transferred bytes attributes also show a very high variance.

Furthermore, Figure 4.6 shows the correlation matrix for the dataset. One can see that the first attributes are very uncorrelated. However, the traffic features considering the host are highly correlated (i.e. REJ error rate and Service REJ error rate have a correlation $> 98\%$). Therefore, during preprocessing, it may be necessary to eliminate some of these features to prevent unwanted effects from collinearity.

Moreover, Figure 4.7 shows a **log-scaled** histogram of the target variable. The distribution of classes is very skewed, potentially making it difficult for the models to make correct predictions.

When considered in combination with box plots of the attribute values, one can see that the majority of variables follow a very skewed distribution. We hypothesize that this may make it a lot more difficult to find reasonable machine learning models.

4.2 Classical Machine Learning Baseline

In the following, we will discuss the methodology for the ML baseline. All models, preprocessing techniques and model selection techniques are implemented in the Scikit-learn [PVG+11] package.

As already discussed in the Section 4.1, it is necessary to delete some attributes in the datasets to prevent collinearity from affecting model performance. Hence, we delete the 'dew point temperature' in the Seoul dataset and all attributes with a correlation $> 90\%$ in the KDD Cup 1999 dataset.

To account for different dataset characteristics, we experiment with five models, namely linear models (logistic regression and linear regression), k-nearest neighbors (K-NN), support vector machine (SVM), random forest (RF), and gradient boosting (GB). We adopt three different scaling methods for each dataset, namely no scaling, standard

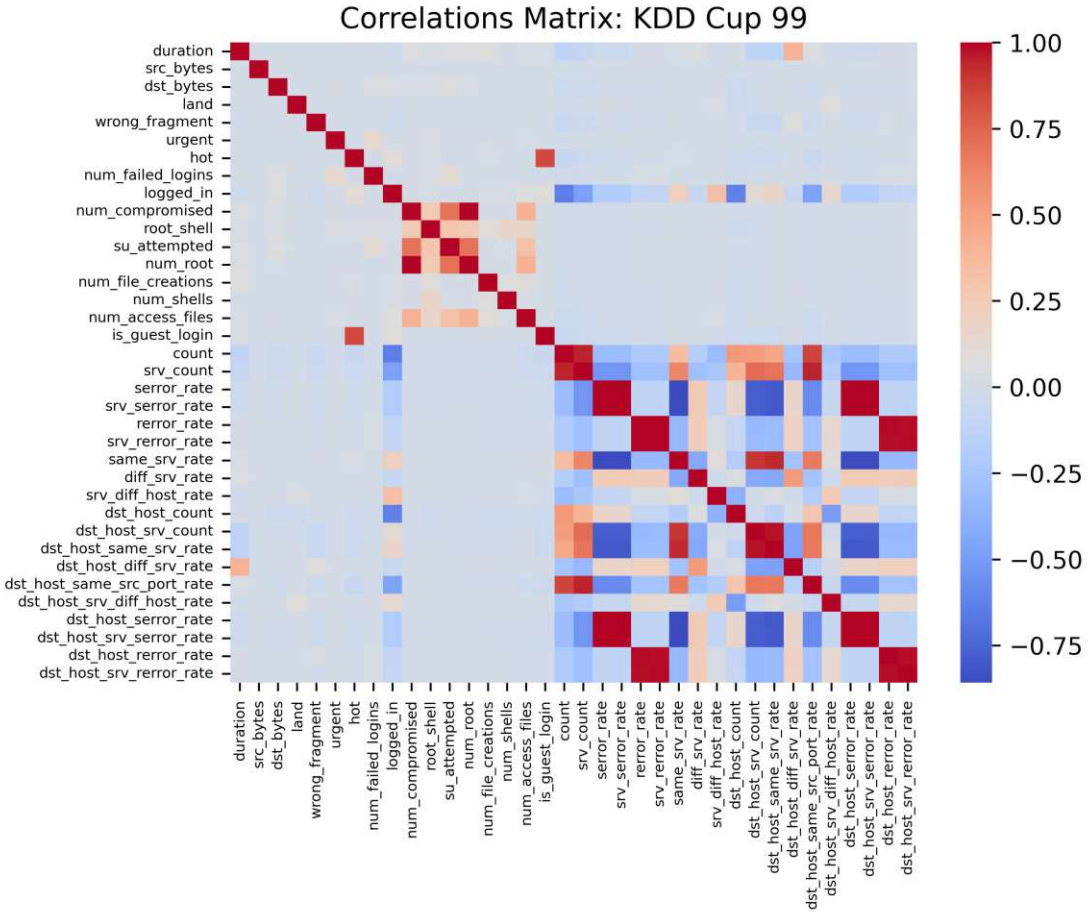


Figure 4.6: KDD Cup 1999 Dataset: Correlations

scaling (remove mean and scale to unit variance), and min-max scaling (scale all values into $[0, 1]$, 0 being the originally smallest and 1 the originally highest value). Furthermore, as the VQA for regression only predicts in range $[-1, 1]$, we scale all regression values to $[0, 1]$ by dividing by the biggest target value in the training set.

We use a grid search cross-validation with 5 folds to optimize the hyperparameters of each model. We choose a subset of 400 samples for hyperparameter optimization and a subset of 250 samples for the evaluation on the validation set, for consistency with the QML experiments. Table 4.5 shows all tested hyperparameters. The hyperparameters do not differ between regression and classification models, except for the linear models.

For each model class, we take the best hyperparameters, which are determined during grid search, to evaluate the performance on the validation set. Based on the validation performance over all model and scaling combinations, we determine the best model. Finally, we train the model again using a subset of 5,000 samples from the training data and evaluate its performance on the held-out test set.

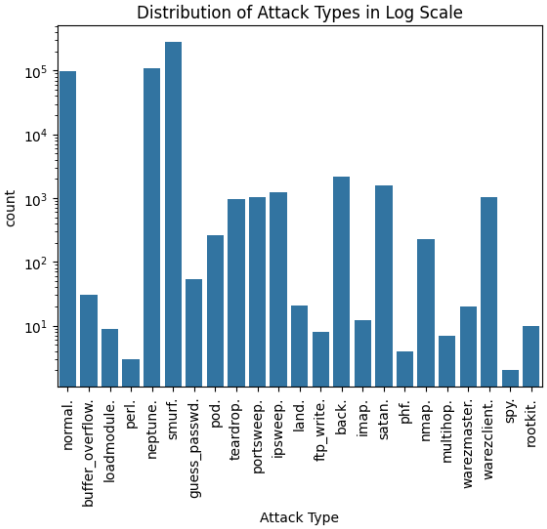


Figure 4.7: KDD Cup 1999 Dataset: Distribution of Target

Model	Hyperparameter	Values
Logistic Regression	penalty C	None, l2 0.01, 0.1, 1, 10
Linear Regression	fit_intercept	True, False
K-Nearest Neighbors	n_neighbors weights algorithm	5, 10, 15, 20, 25, 30 uniform, distance auto, ball_tree, kd_tree, brute
Support Vector Machine	kernel C Gamma Degree	linear, rbf, poly 0.1, 1, 10 0.01, 0.001 (rbf) 2, 3 (poly)
Random Forest	n_estimators max_features max_depth	50, 100, 200 sqrt, log2 4, 6, 8
Gradient Boosting	n_estimators max_features max_depth criterion	50, 100, 200 sqrt, log2 4, 6, 8 friedman_mse, squared_error

Table 4.5: Classical Machine Learning: Hyperparameters

For regression, we use the mean squared error as the criterion and the mean absolute error as an additional performance metric. For the classification task, we use the weighted f-1 score as the criterion and the accuracy as an additional measure. Furthermore, we measure the time for hyperparameter optimization, fitting of the final model and the inference time on the test set.

4.3 Quantum Machine Learning

Optimizing hyperparameters is an essential step in building effective ML models. However, it becomes especially challenging when working with complex model classes like VQAs. These algorithms have various feature encodings, ansatzes, and optimizers, each with many hyperparameters that require tuning. As a result, providing a comprehensive evaluation of all possible combinations of hyperparameters can be a daunting and time-consuming task.

To address this challenge, we choose only a subset of hyperparameters. However, even with the reduced set of configurations, the optimization process is still computationally intensive. The models run longer than their classical counterparts, as we will show in Section 5, which further limits our ability to extensively explore hyperparameters. Still, while the optimization process can be cumbersome, we take steps to evaluate a reasonable amount of configurations that are, presumably, likely to provide valuable insights into the performance of VQAs for our specific use case.

We use the preprocessing techniques as they are implemented in the Scikit-learn [PVG+11] package. Furthermore, we use the open source package Qiskit [con23] for the QML components.

4.3.1 Preprocessing

To prepare our data for training the VQAs, we follow a similar preprocessing approach as the classical baseline. We begin by removing highly correlated features, where necessary, to prevent collinearity of affecting the performance of our models. However, since we are dealing with quantum hardware, we have to limit our experiments to using only seven qubits. This restriction is crucial, as we want to be able to run and evaluate the final models on real quantum hardware, and IBM's free plan only allows for the use of five or seven qubit machines. Therefore, we employ the popular approach of using principal component analysis (PCA).

PCA works by creating new features using a linear combination of the existing ones, while keeping most of the variance contained in the data. This technique is unsupervised, meaning that the labels are not used for feature reduction. It is important to scale the data appropriately, as PCA uses distances between data points to create new features [RRC19].

Apart from PCA, we also experiment with another preprocessing technique called linear discriminant analysis (LDA), which is only relevant for classification. We discovered

that LDA outperformed PCA as a preprocessing step in two previous studies [MP22, HQBCTC22]. Unlike PCA, LDA is a supervised technique, meaning that the labels are used to reduce the dimensionality of the data. The goal of LDA is to obtain a representation of the data where points belonging to different classes are far apart, while points of the same class are clustered together [TGIH17].

Unfortunately, the Qiskit implementation of a VQA for regression only predicts in range $[-1, 1]$. We therefore divide by the biggest value in the training set to scale our target into $[0, 1]$.

4.3.2 Feature Encoding

As discussed in Section 2.3.3, the feature mapping proposed in [HCT+19] exhibits desirable properties because it is hard to reproduce classically and space efficient. It is therefore widely used in QML and will be used for our experiments as well.

Qiskit implements the feature map as the PauliFeatureMap[7], which allows the usage of (combinations of) the Pauli matrices I, X, Y, Z. From the extended version, two subclasses, the ZFeatureMap[8] and the ZZFeatureMap[9] are derived. The ZFeatureMap sets $k = 1$ and $P_0 = Z$, hence not entangling the qubits, and the ZZFeatureMap sets $k = 2$, $P_0 = Z$ and $P_{0,1} = ZZ$. We experiment with these two instantiations, given that it is possible to use them out-of-the-box without running extensive experiments tuning the Pauli matrices. Furthermore, we believe it is interesting to compare them, as ZFeatureMap does not entangle the qubits, whereas ZZFeatureMap does.

For the ZZFeatureMap, we vary the entanglement strategy used by the entangling blocks. The entanglement options we choose are 'full', 'linear', 'sca', 'pairwise', and 'circular', as explained in the Ansatzes section of Section 2.3.4. We decide to leave out 'reverse linear', due to the similarity to 'linear'. We do not perform hyperparameter tuning for the ZFeatureMap.

4.3.3 Ansatz

Regarding the ansatzes, we implement four popular approaches: PauliTwoDesign, EfficientSU2, RealAmplitudes, and TwoLocal, as discussed in Section 2.3.4. Each ansatz has different hyperparameters that can be optimized to improve the performance of the quantum circuit.

For the PauliTwoDesign ansatz, we only set the seed to a fixed number to make the experiments more reproducible. RealAmplitudes, on the other hand, requires optimization

[7] https://qiskit.org/documentation/stable/0.24/stubs/qiskit.circuit.library.PauliFeatureMap.html. Accessed 24.04.2023

[8] https://qiskit.org/documentation/stable/0.24/stubs/qiskit.circuit.library.ZFeatureMap.html. Accessed 24.04.2023

[9] https://qiskit.org/documentation/stable/0.24/stubs/qiskit.circuit.library.ZZFeatureMap.html. Accessed 24.04.2023

Dataset	Optimizer	Iterations Plain	Iterations Noise
KDD	COBYLA	500	250
KDD	SPSA	250	125
KDD	Nelder-Mead	250	100
Covertype	COBYLA	500	250
Covertype	SPSA	300	125
Covertype	Nelder-Mead	150	75
California Housing	COBYLA	500	250
California Housing	SPSA	250	125
California Housing	Nelder-Mead	250	100
Seoul Bike Sharing	COBYLA	500	250
Seoul Bike Sharing	SPSA	250	125
Seoul Bike Sharing	Nelder-Mead	250	100

Table 4.6: Iterations for Optimizer

of the entanglement parameter, which has the same options as the ZZFeatureMap does, except for 'pairwise', which is not supported. We will not be tuning other hyperparameters for this ansatz.

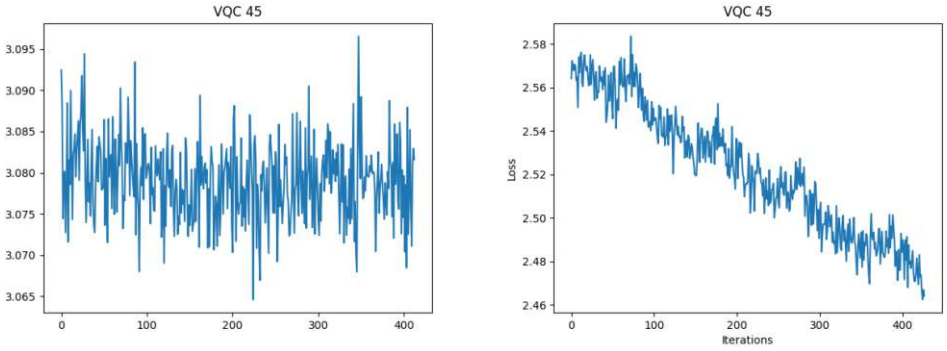
EfficientSU2 also supports the entanglement parameter, with the same options as RealAmplitudes. We leave the other hyperparameters to their default values. Finally, for the TwoLocal ansatz, we experiment with the entanglement parameter as well. This ansatz supports the same strategies as the ZZFeatureMap does. Moreover, the rotation and entanglement blocks need to be set here. We chose 'ry' as the rotation block and 'cx' as the entanglement block, which is a recommended setting in the documentation.

4.3.4 Optimizer

In Section 2.3.4, we discussed three different classical optimization algorithms that we used for our experiments: COBYLA, SPSA and Nelder-Mead. For each dataset and optimizer combination, we do some exploratory checking to see how many iterations it would take to converge. We do so by plotting the loss in every iteration step. Furthermore, running the simulators with noise models increases the training time drastically, hence, we choose a separate iteration count, depending on the convergence of the models. Table 4.6 shows the iteration count for all configurations.

Furthermore, we set a tolerance value of 0.1. Both COBYLA and Nelder-Mead support the parameter out-of-the-box, unfortunately SPSA does not.

Other than that, we leave the hyperparameters to their default values for COBYLA and SPSA. For Nelder-Mead, we set the adaptive parameter to 'True', which adapts the hyperparameters specifically for the dimensionality of the problem. In the beginning, we found that a lot of times, the loss function did not change at all, as can be seen in



(a) Nelder-Mead Non-Adaptive

(b) Nelder-Mead Adaptive

Figure 4.8a. After changing it, some configurations improved (i.e., Figure 4.8b), we did not see great improvements, however.

4.3.5 Noise Models

For the noise models, we use noise models from real machines, which are available through Qiskit. There are four different 7-qubit machines to be considered, all of them have the same basis gates, instructions that are noisy, qubits with noise (all of them) and the specific qubit errors. An example of a noise model is shown in the following.

```
<IBMBackend('ibm_perth')> NoiseModel:
  Basis gates: ['cx', 'delay', 'id', 'if_else', 'measure',
    'reset', 'rz', 'sx', 'x']
  Instructions with noise: ['id', 'x', 'measure', 'reset',
    'sx', 'cx']
  Qubits with noise: [0, 1, 2, 3, 4, 5, 6]
  Specific qubit errors: [
    ('id', (0,)), ('id', (1,)), ('id', (2,)), ('id', (3,)),
        ('id', (4,)), ('id', (5,)),('id', (6,)),
    ('sx', (0,)), ('sx', (1,)), ('sx', (2,)), ('sx', (3,)),
        ('sx', (4,)), ('sx', (5,)), ('sx', (6,)),
    ('x', (0,)), ('x', (1,)), ('x', (2,)), ('x', (3,)),
        ('x', (4,)), ('x', (5,)), ('x', (6,)),
    ('cx', (6, 5)), ('cx', (5, 6)), ('cx', (4, 5)),
        ('cx', (5, 4)), ('cx', (3, 5)), ('cx', (5, 3)),
        ('cx', (3, 1)), ('cx', (1, 3)),('cx', (2, 1)),
        ('cx', (1, 2)), ('cx', (0, 1)), ('cx', (1, 0)),
    ('reset', (0,)), ('reset', (1,)), ('reset', (2,)),
        ('reset', (3,)), ('reset', (4,)),
        ('reset', (5,)), ('reset', (6,)),
```

Step	Object	Configurations
Preprocessing	PCA	-
	LDA	-
Feature Encoding	ZZFeatureMap	'ent.':['full', 'linear', 'sca', 'pairwise', 'circular']
	ZFeatureMap	-
Ansatz	PauliTwoDesign	-
	EfficientSU2	'ent.':['full', 'linear', 'sca', 'circular']
	RealAmplitudes	'ent.':['full', 'linear', 'sca', 'circular']
	TwoLocal	'ent.':['full', 'linear', 'sca', 'pairwise', 'circular']
Optimizer	COBYLA	'maxiter', 'tol'=0.1
	SPSPA	'maxiter'
	Nelder-Mead	'maxiter', 'tol'=0.1

Table 4.7: Summary: QML Hyperparameters
Abbrev.: ent. = entanglement

```
('measure', (0,)), ('measure', (1,)), ('measure', (2,)),
    ('measure', (3,)), ('measure', (4,)),
    ('measure', (5,)), ('measure', (6,))]
```

4.3.6 Hyperparameter Tuning

Table 4.7 shows a summary of all hyperparameter configurations we test. In total, they sum up to 168 configurations for classification per optimizer, i.e., 672 configurations per dataset, and 84 configurations for regression per optimizer.

We preprocess the data in the same way as for the baseline and fit the PCA (applying a StandardScaler before) and the LDA, the latter only if we employ classification. Afterward, we sample 400 samples for hyperparameter tuning. Due to the extensive runtime, we cannot consider more samples. For validation, we take a subset of 250 samples from the validation set.

Then, we start an exhaustive search for all configurations, by creating and training the models. We then calculate the accuracy and f1 score for classification, or the MSE and MAE for regression on the validation set. Finally, we take the best model, according to the validation performance, and retrain the model again using a larger dataset of 5,000 samples (again from the training data). We evaluate the test performance on the held-out test set.

CHAPTER 5

Results

In the following, we will present the results from our experiments. In particular, we will first explain our experimental setup in Section 5.1 to set the reported running times into perspective.

Then, in Section 5.2, we will discuss the results for the classical ML baseline for every dataset separately. In Section 5.3 we will do the same for the QML models.

Afterward, in Section 5.4, we will provide an extensive evaluation and comparison between the two.

5.1 Experimental Setup

The classical baseline experiments are conducted on the following Linux-based computer:

- **Operating System**: Fedora Linux 38
- **CPU**: Intel(R) Xeon(R) W-2123 CPU (8 cores @ 3.60 GHz)
- **Memory**: 64 GB

Furthermore, we use a Linux-based computer with the following configuration for the QML experiments.

- **Operating System**: Debian/GNU Linux 11
- **CPU**: Intel(R) Xeon(R) CPU E5-2623 v4 (16 cores @ 2.60GHz)
- **Memory**: 128 GB

IBM offers quantum computers with up to seven qubits and quantum simulators in the cloud through their 'Open Plan' and their open source Python package Qiskit [con23]. Furthermore, through Qiskit Aer it is possible to run simulators with and without noise locally. The noise models can be custom-built, as well as derived from actual quantum machines. The three possibilities can be used interchangeably by changing only the backend in the code. To offer the possibility of running the experiments on real quantum hardware with minimal additional effort, we decide to use Qiskit for our experiments. Furthermore, all relevant QML models and components are conveniently implemented in the package.

We use the following packages and versions.

- **Python**: 3.11.4
- **Qiskit**: 0.42.1
- **Scikit-learn**: 1.2.2

5.2 Classical Machine Learning Baseline

Table 5.1 gives an overview of the results we achieved on the different datasets. In particular, it lists the dataset, best model and scaling, metric and achieved values for the metric.

It can be seen that the predictions for the California Housing dataset are very accurate according to their MSE. The same holds true for the Seoul bike sharing dataset. However, as already discussed in Section 4.1, the variance, especially for the afternoon hours, is very high, which makes it difficult to make reasonable predictions there. We will discuss the results in more detail later on to compare the prediction accuracy to the hour attribute.

Looking at the classification datasets, we can see that there is hardly any difference between the f-1 scores and the accuracy. For the cover type dataset, we achieve only a 71% accuracy. We believe a pitfall was tuning using 400 samples only, leading to difficulties producing models that generalize well to unseen data. Furthermore, for the KDD Cup dataset, the final model achieves a 99% accuracy.

5.2.1 California Housing Dataset

We obtain a MSE of 0.0105 and a MAE of 0.0691 for the final model. GB models often work great on a variety of problems, hence, we are not surprised that the best model was a gradient boosted one.

Initially, we want to analyze where big residuals in the predictions occur. Therefore, we plot the predicted vs. the actual values in Figure 5.1, the x-axis showing the true values and the y-axis the predictions. Unfortunately, the model predicts the same values for almost all data points. The plot shows that, given the input data, it is favorable and

Dataset	Model	Measurement	Value
California Housing	Gradient Boosting, MinMax Scaling	MSE MAE	0.0105 0.0691
Seoul Bike Sharing	Gradient Boosting	MSE MAE	0.0063 0.0539
Cover Type	Support Vector Machine, Standard Scaling	F-1 Accuracy	0.71 0.71
KDD Cup	Random Forest	F-1 Accuracy	0.99 0.99

Table 5.1: Classical Machine Learning: Results

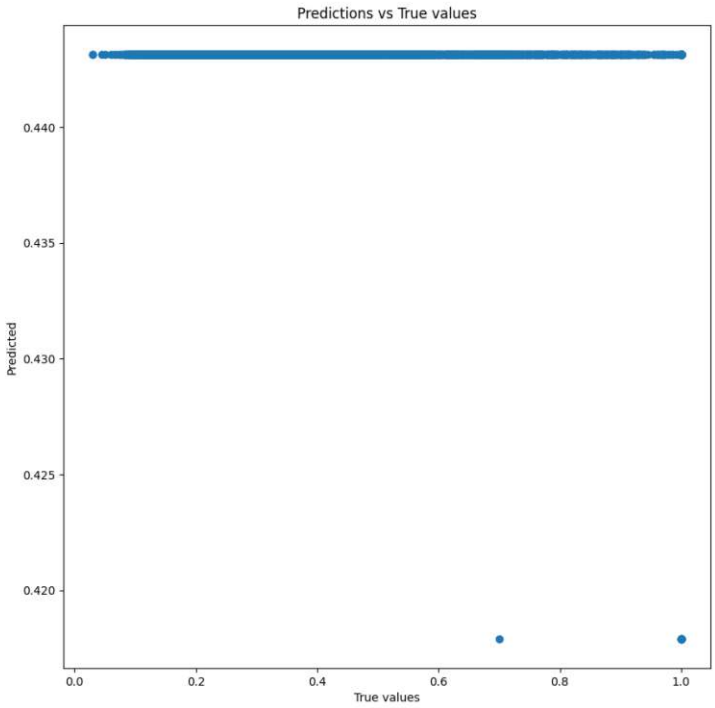


Figure 5.1: Baseline California Housing: Predicted vs. Actual

leads to the lowest MSE when the model predicts approximately the mean every time. We believe that training using only a small subset could be the reason for that.

Furthermore, we summarize all grid search configurations in Appendix C in Table C.1. Ensemble models work well for this dataset, with GB and RF scoring best. Many estimators are favorable, the optimal number for GB was 300, for the RF it was 200 or

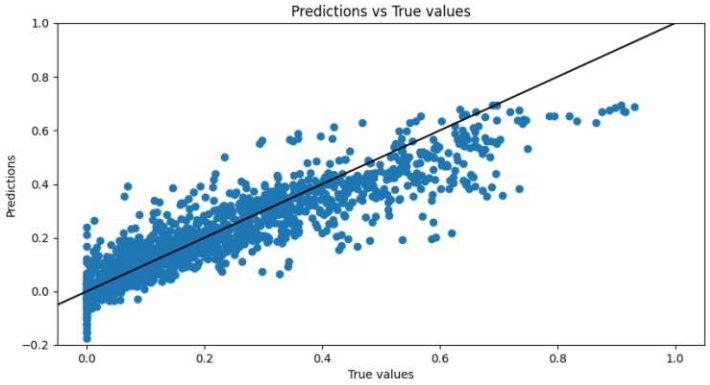


Figure 5.2: Baseline Seoul Bike Sharing: Predicted vs. Actual

250. Nonetheless, linear regression and SVM did not show big performance differences either. Another thing that stands out is the excessive grid search tuning time for the SVM with no scaling, as compared to standard and minmax scaling, even though the same parameters are tested.

5.2.2 Seoul Bike Sharing Dataset

We achieve a MSE of 0.0063 and a MAE of 0.0539 on the Seoul Bike sharing dataset. We choose the same approach as for the California housing dataset to analyze the predictions for the bike sharing dataset further. Figure 5.2 shows the predicted vs. actual values, with the black line showing the optimal predictions.

One thing that stands out, is that the model predicts negative values. Interestingly enough, most of the strongly negative values have a true value of 0 or close to zero, hence, setting all negative values to zero automatically (which is reasonable as a negative demand does not make any sense at all), could improve the results. By doing so, we can reduce the MSE to 0.0061, representing a decrease of 3%, and the MAE to 0.0518, representing a decrease of 4%.

Furthermore, we can look at the prediction accuracy per hour. We plot the residuals vs. the hour attribute in Figure 5.3. As hypothesized earlier, the hours with the greatest standard deviation (7-9 and 17-19) are the most challenging to predict. Furthermore, the afternoon demand is more difficult to predict than the forenoon demand. There are discrepancies between the predictive performances of different hours.

The grid search results are shown in Appendix C in Table C.2. Similarly to the results of the California housing dataset (see Section 5.2.1), ensemble models again outperform all other ones. Many estimators for the models are favorable (300 and 250 for GB and RF respectively). Interestingly, a simple K-NN regressor with 10 neighbors for MinMax or Standard Scaling returns quite good results already.

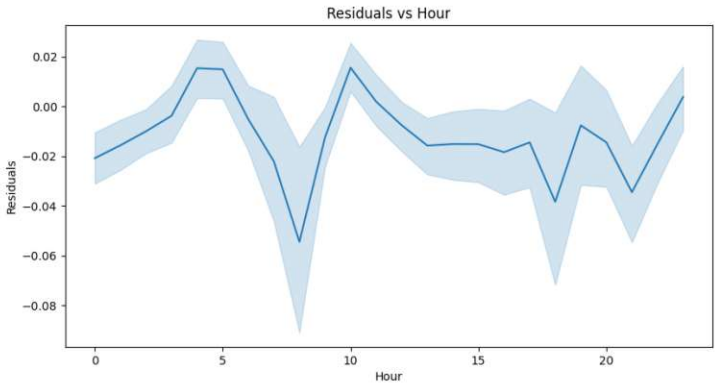


Figure 5.3: Baseline Seoul Bike Sharing: Residual vs. Hour

5.2.3 Cover Type Dataset

The final models for the cover type dataset achieve an f-1 score and accuracy of only 71%. In the following, we will investigate the results further to see where potential improvement could be made.

Confusion matrices, which we adopt for our experiments, are popular tools to analyze the results of classification tasks further. The confusion matrix for the cover type dataset can be seen in Figure 5.4. In particular, the x-axis shows the predicted labels, whereas the y-axis shows the true ones and the matrix is normalized by row. They allow analyzing which classes are often confused with each other and which classes are particularly easy/difficult to predict.

We can see that the classifier often predicts `spruce/fir` or `lodgepole pine`, which are the classes that are most often represented in the dataset. The predictive performance for class `cottonwood/willow` is a lot lower than for other classes, which could be due to the class having the least samples in the training set. With accuracies $\geq 70\%$, the predictive performance for `spruce/fir`, `lodgepole pine` and `ponderosa pine` are good, however, it fails to correctly predict the other classes.

The difficulties in predicting the different classes may be due to an imbalanced dataset, plus using only 5,000 samples for the final training. For a comparison of the distributions in the different datasets, please refer to Figure 5.5. It can be seen that `spruce/fir`, and `lodgepole pine` make up the majority of samples, whereas `aspen` and `cottonwood/willow` are underrepresented. Still, the distribution of classes is mostly kept in our training, validation and test sets.

The grid search results are again shown in Appendix C in Table C.3. The validation accuracies hardly differ from the final test performance, which is interesting, given that we trained and evaluated only on a small subset of data. It can be seen that tuning the SVM takes substantially longer without scaling than with. SVM with standard scaling outperforms other models, interestingly, also the relatively simple Logistic Regression

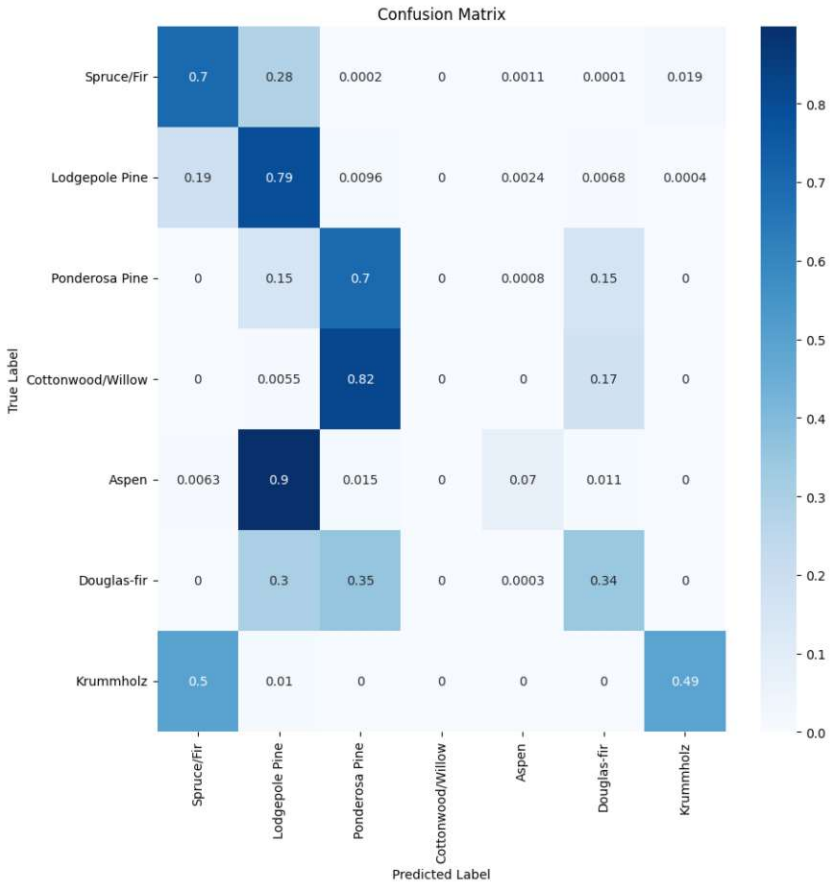


Figure 5.4: Baseline Cover Type: Confusion Matrix

with MinMax scaling performs comparable to the best model. K-NN seems to not be able to capture meaningful connections. Many estimators are again favorable for the GB (200 or 250).

5.2.4 KDD Cup 1999 Dataset

Finally, we achieve a 99% accuracy for the KDD Cup dataset. Still, in the following, we investigate which classes were easy to predict and which were hard. One thing that stands out is that even relatively simple logistic regression or K-NN models are already able to perform that well on the dataset (with a 97% validation accuracy).

We choose to plot confusion matrices for the KDD Cup predictions as well, which can be seen in Figure 5.6. While some attacks (`back` `neptune`, `smurf`, `teardrop`) have an almost perfect accuracy. the most dangerous misclassifications are those that are misclassified as being `normal`. Unfortunately, almost all `buffer overflow` and `imap` attacks are classified as being `normal`. Also, a big fraction of `guess password`, `nmap` and `warezclient` attacks are classified as being `normal`.

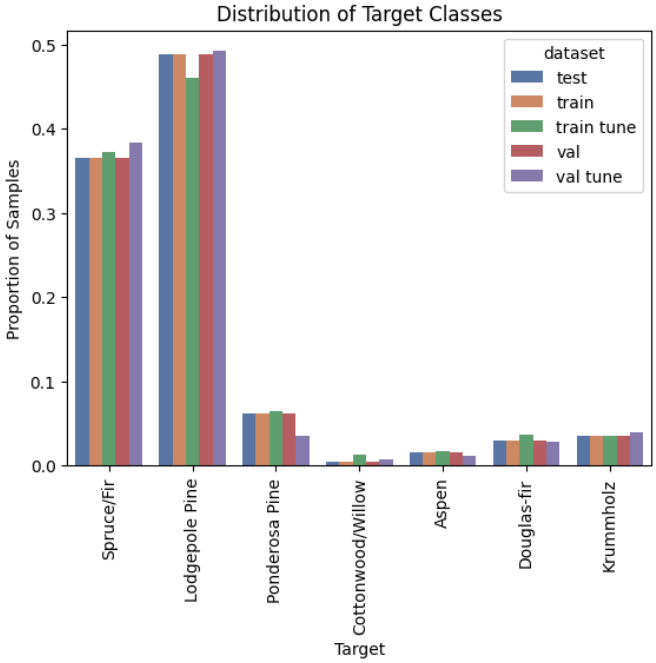


Figure 5.5: Target Distribution Cover Type

We show the results of the grid search in Appendix C in Table C.4. There is no one best performing model, as RF and GB all achieve a 98% accuracy and all others either 96% or 97%. The results are in particular interesting, as we get good validation performances, even though tuning with only a subset again. The dataset seems to allow good predictions based on only few samples already.

5.3 Quantum Machine Learning

In the following, we will present the results we obtained on the four datasets. In particular, for each dataset, we will discuss the results of the hyperparameter optimization with and without noise and the final performance of the best model.

We give an overview of the final performances of our models in Table 5.2. There are big discrepancies between the performances of the classical machine learning models and the quantum ones, which we will dive deeper into in the following analysis.

When we refer to the best configurations in the following, we mean all configurations that are within 10% accuracy (classification) or 10% span (max-min) of the MSE (regression) of the best configuration. The worst configurations are similarly defined. They are within 10% accuracy and 10% span of the MSE of the worst configuration.

Furthermore, we perform significance tests with a significance level of 5%. We choose non-parametric tests. In particular, we choose a Friedman Test for dependent distributions of

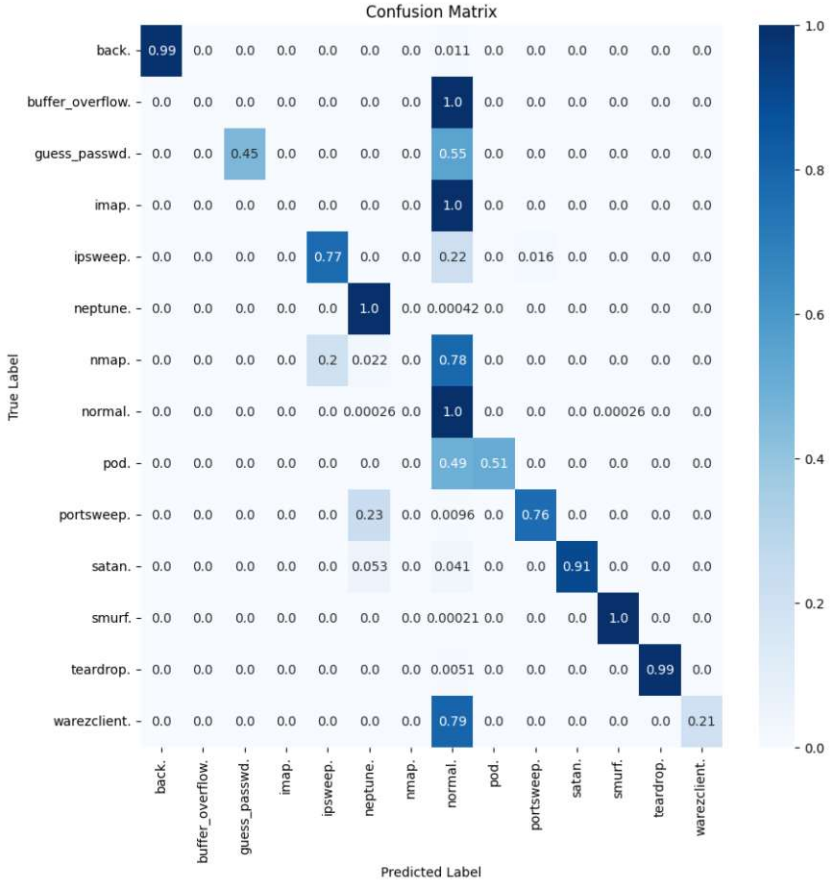


Figure 5.6: Baseline KDD Cup 1999: Confusion Matrix

more than two groups and the Wilcoxon Signed Rank test for comparing two dependent distributions. Moreover, we use the Kruskal-Wallis test for independent distributions of more than two groups and the Mann-Whitney-U test for independent distributions of two groups.

In addition to comparing the experiments with and without noise separately, we compare each configuration with and without noise as well. We perform paired Wilcoxon tests to see if they differ significantly. Whenever we refer to 'differences' in this context, we talk about the mean absolute difference in performance for the configurations with and without noise.

Whenever we provide convergence plots, the values on the x-axis do not represent the actual iterations for SPSA and Nelder-Mead. The methods 'probe' candidate solutions and afterward decide to (1) take it as the next candidate or (2) go back and try a different one. In the callback function, we do not have access to whether or not the step is accepted or not, therefore, we plot the whole sequence. Nevertheless, the graphs show whether the loss converges.

Dataset	Model	Metric	Value
California Housing	SPSA, ZFeatureMap, EfficientSU2 - full	MSE MAE	0.1315 0.2798
California Housing with Noise	COBYLA, ZFeatureMap, EfficientSU2 - full	MSE MAE	0.1141 0.2543
Seoul Bike Sharing	SPSA, ZFeatureMap, EfficienSU2 - circular	MSE MAE	0.0467 0.1627
Seoul Bike Sharing with Noise	SPSA, ZFeatureMap, RealAmplitudes - circular	MSE MAE	0.0547 0.1687
Cover Type	COBLYA, ZFeatureMap, TwoLocal - pairwise, LDA	F-1 Accuracy	0.5808 0.5957
Cover Type with Noise	COBYLA, ZFeatureMap, RealAmplitudes - circular, LDA	F-1 Accuracy	0.5581 0.5828
KDD Cup	SPSA, ZFeatureMap, TwoLocal - circular, PCA	F-1 Accuracy	0.5765 0.5432
KDD Cup with Noise	COBYLA, ZFeatureMap, TwoLocal - circular, PCA	F-1 Accuracy	0.5589 0.5748

Table 5.2: Quantum Machine Learning: Results

5.3.1 California Housing Dataset

We will now present the results for the California housing dataset, first on a perfect simulator, then with noise and, finally, we will compare the two and point out significant differences.

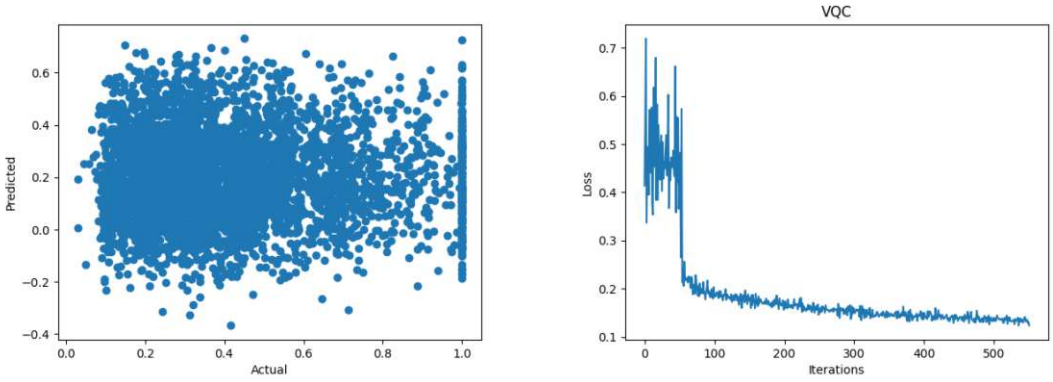
5.3.1.1 Noiseless Configuration

We find big discrepancies between the test performance of the final model and the performance of the best model during hyperparameter optimization. The final test MSE is at 0.1315 and the MAE at 0.2798. Looking at the predicted vs. actual plot in Figure 5.7a, reveals that the model fails to find reasonable patterns.

Interestingly, while the convergence is very smooth, as can be seen in Figure 5.7b, the final training loss is still quite high, at an MSE of about 0.1. Nonetheless, the hyperparameter tuning experiments give us valuable insights, which we will discuss in the following.

Table 5.3 lists the ansatz, optimizer, feature map, ansatz entanglement, MSE, MAE and time for the top five configurations during hyperparameter tuning. It can be observed that EfficientSU2 ansatz configurations are very common, and the majority uses SPSA as the optimizer. Furthermore, all configurations use the ZFeatureMap.

Furthermore, in Appendix D in Table D.1, we list the top 40 configurations. The MSE values range from 0.04-0.3, so we can observe significant differences depending on the



(a) Predicted vs. Actual (b) Convergence

Figure 5.7: QML California: Final Predictions

Ansatz	Optimizer	FeatMap	Ans.Ent.	MSE	MAE	Time
EfficientSU2	SPSA	Z	full	0.0460	0.1758	31843s
EfficientSU2	SPSA	Z	sca	0.0511	0.1757	26506s
EfficientSU2	COBYLA	Z	circular	0.1774	0.1774	15496s
RealAmplitudes	SPSA	Z	sca	0.0546	0.1854	21318s
EfficientSU2	SPSA	Z	circular	0.0558	0.1939	26568s

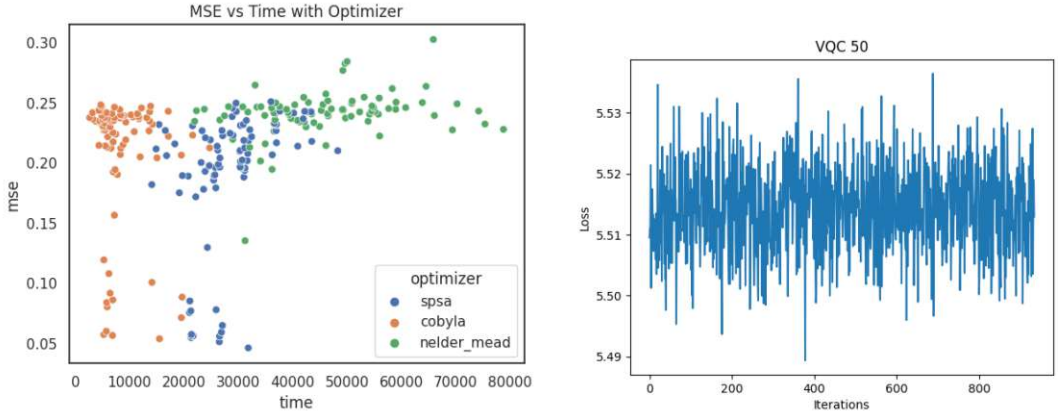
Table 5.3: QML California: Top 5 Configurations

configurations. We will present our observations for every parameter separately in the following.

Optimizer We can observe significant differences depending on the optimizer used. In particular, Nelder-Mead, with a mean MSE of 0.24, is outperformed by both COBYLA (0.20) and SPSA (0.18). Furthermore, SPSA performs significantly better than COBYLA. Still, the set of best configurations (within 10% span of the best one) are made up of five COBYLA and eight SPSA configurations, so there are several very good COBYLA configurations as well. Figure 5.8a visualizes the performance of different optimizers. It can be seen that some configurations from both COBYLA and SPSA work really well, however, usually, the SPSA configurations are slightly better.

While there are usually some configurations for the optimizers that return above-average results, such behavior cannot be observed for Nelder-Mead. Looking at the convergence plots, such as Figure 5.8b, reveals that the optimizer has troubles finding any useful optimization path.

Ansatz There are significant differences between the ansatzes. In particular, we find RealAmplitudes (mean MSE 0.20) and TwoLocal (0.20) to be significantly better than PauliTwoDesign (0.22). We can observe no significant differences for EfficientSU2, with a



(a) Optimizer (b) Nelder-Mead Convergence

Figure 5.8: QML California: Optimization

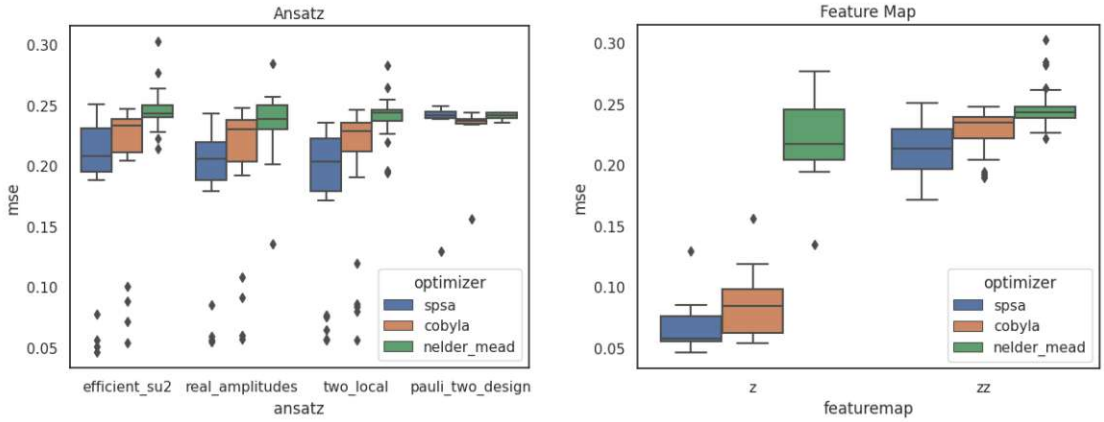
mean MSE of 0.21. Nonetheless, EfficientSU2, TwoLocal and RealAmplitudes are equally represented among the set of best performing configurations. Figure 5.9a compares them visually. It can be seen that the ansatzes using Nelder-Mead as the optimizer perform equally bad, however, EfficentSU2, TwoLocal and RealAmplitudes all have some good configurations for the other optimizers. The range of PauliTwoDesign configurations is very narrow all in all.

Ansatz Entanglement We find no significant differences between the ansatz entanglement strategies, neither when considering them on their own, nor in combination with the different ansatzes. Nevertheless, only 'circular', 'full' and 'sca' entanglement configurations are among the best-performing ones.

Feature Map ZFeatureMap significantly outperforms ZZFeatureMap with a mean MSE of 0.12 and 0.22 respectively. All best configurations use the ZFeatureMap and the performances are compared in Figure 5.9b. It can be seen that almost all ZFeatureMap configurations for SPSA and COBYLA outperform the ZZFeatureMap ones, however, the advantage cannot be observed for Nelder-Mead.

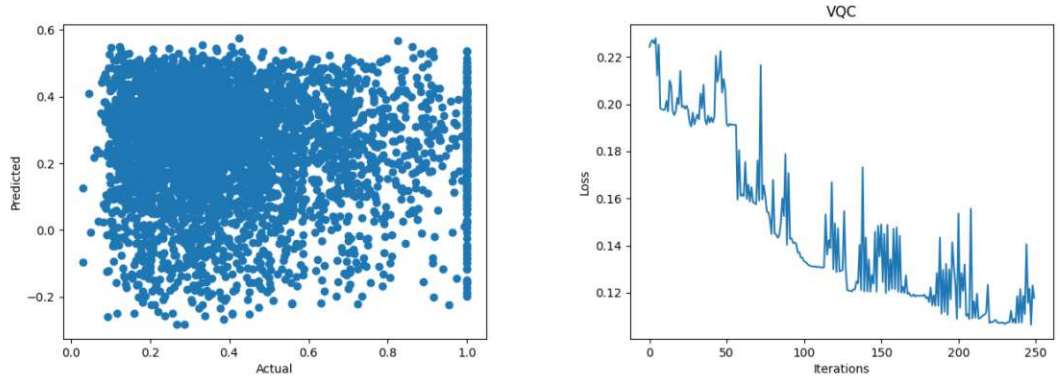
Feature Map Entanglement When we compare the feature map entanglement, we find that no entanglement (ZFeatureMap) significantly outperforms all other entanglement strategies. Furthermore, both 'linear' and 'pairwise' entanglement outperform 'circular', 'full' and 'sca'.

Moreover, we can analyze which configurations make up the worst-performing ones. We find only three configurations within 10% span of the worst configuration, and all of them use Nelder-Mead as the optimizer and ZZFeatureMap as the feature map. There is one configuration for each of the ansatzes except PauliTwoDesign, and all of them use 'full' as the entanglement strategy.



(a) Ansatz (b) Feature Map

Figure 5.9: QML California: Ansatz and Feature Map



(a) Predicted vs. Actual (b) Convergence

Figure 5.10: QML California with Noise: Final Predictions

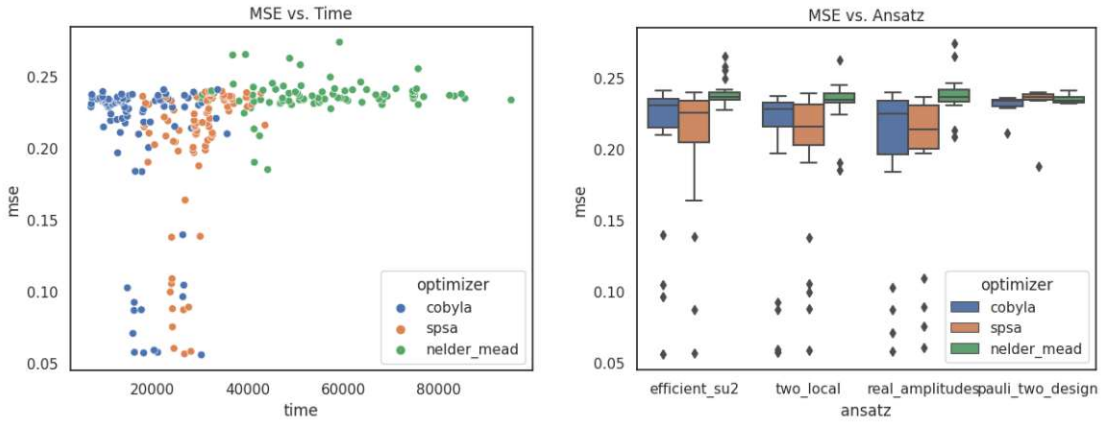
5.3.1.2 Noisy Configuration

The results with noise are similar to the ones without noise, with a MSE of 0.1141 and MAE of 0.2543. The prediction plot, in Figure 5.10a, looks very similar. Interestingly, the convergence plot shows the optimization happens in stages, with only small changes for some time, followed by bigger drops in loss repeatedly. Unfortunately, it converges quite early at a training loss of about 0.11. We hypothesize that it got stuck in a barren plateau (BP).

Table 5.4 lists the ansatz, optimizer, feature map, ansatz entanglement, MSE, MAE and time for the top five configurations. Furthermore, in Appendix D in Table D.2, we list the top 40 configurations. The MSE ranges from 0.05 to 0.23. Interestingly, while the best configurations are slightly worse than when running the experiments without noise,

Ansatz	Optimizer	Feature Map	Entang.	MSE	MAE	Time
EfficientSU2	COBYLA	Z	full	0.0558	0.1792	30269s
EfficientSU2	SPSA	Z	sca	0.0566	0.1906	26742s
TwoLocal	COBYLA	Z	sca	0.0573	0.1803	18172s
TwoLocal	COBYLA	Z	full	0.0576	0.1846	21107s
RealAmplitudes	COBYLA	Z	full	0.0576	0.1820	16235s

Table 5.4: QML California with Noise: Top 5 Configurations



(a) Optimizer (b) Ansatz

Figure 5.11: QML California with Noise: Optimization and Ansatz

the performances of the worst solutions actually increase.

Optimizer We can observe significant differences depending on the optimizer used. In particular, Nelder-Mead, with a mean MSE of 0.23, is significantly outperformed by both COBYLA (mean MSE 0.20) and SPSA (0.20). There are no significant differences between COBYLA and SPSA, and they are equally represented among the best configurations. Figure 5.8a visualizes the performance of different optimizers. It can be seen that some configurations from both COBYLA and SPSA can be found among the best ones, however, Nelder-Mead ones consistently underperform. The same convergence behavior as has been reported before (see Figure 5.27) can be found for Nelder-Mead.

Ansatz We find no significant differences between the ansatzes. Figure 5.9a compares them. All of them perform poorly when using Nelder-Mead and on average for the other ansatzes, however, there are several positive outliers for EfficientSU2, TwoLocal and RealAmplitudes. PauliTwoDesign configurations show no such behavior. In addition to that, they perform worse in mean than the other ansatzes and no PauliTwoDesign configurations are among the best performing ones.

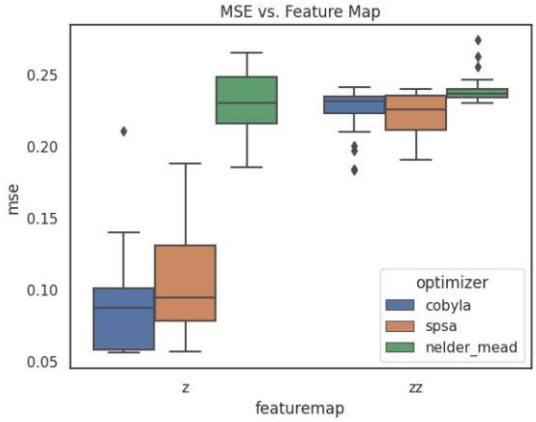


Figure 5.12: QML California with Noise: Feature Map

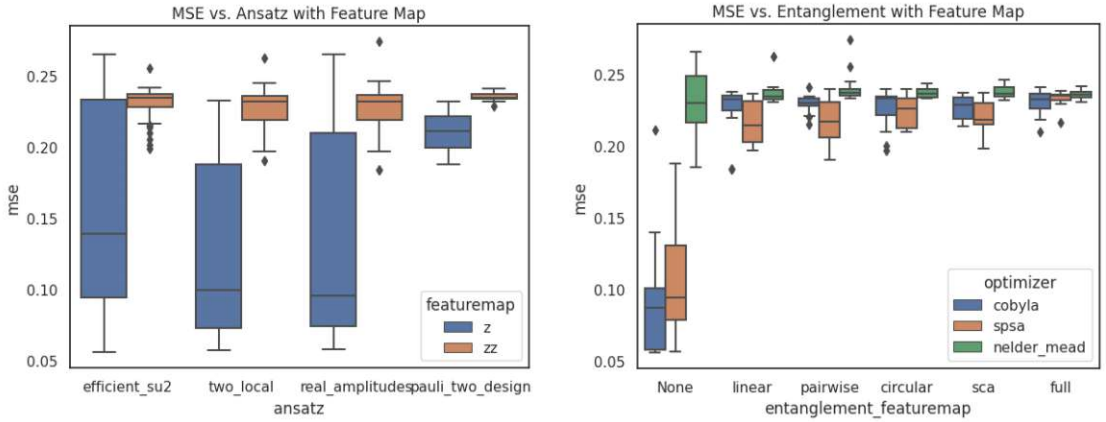
Ansatz Entanglement Similarly, no significant differences can be observed regarding the ansatz entanglement strategies. However, only 'full', 'sca' and 'circular' entanglement configurations can be found among the best performing ones. When we consider the ansatz combined with the entanglement, we find no significant differences for any ansatz.

Feature Map ZFeatureMap significantly outperforms ZZFeatureMap with a mean MSE of 0.14 and 0.22 respectively. All best configurations use the ZFeatureMap. Figure 5.12 compares the performances. It can be seen that almost all ZFeatureMap configurations using SPSA and COBYLA outperform the ZZFeatureMap ones, however, the advantage cannot be observed for Nelder-Mead.

Feature Map Entanglement When we compare the feature map entanglement, we find that no entanglement (ZFeatureMap) significantly outperforms all other entanglement strategies. Furthermore, 'linear' and 'sca' entanglement outperform 'full'.

Moreover, we analyze the set of worst-performing configurations. Only Nelder-Mead configurations can be found within the set of worst configurations. Interestingly, we find configurations for all ansatzes except PauliTwoDesign, insinuating that the ansatz has a relatively stable performance with little deviations. Interestingly, the ratio of ZFeatureMap configurations (7% out of all) in the set is a lot higher than for ZZFeatureMap (1%), implying a similar behavior than the PauliTwoDesign ansatz. Only 'circular', 'full' and 'sca' ansatz entanglement configurations can be found in the set, while we can only find 'linear' and 'pairwise' feature map entanglement strategies there.

Furthermore, we look at the performances of feature map and ansatz combinations in Figure 5.13a. It can be seen that ZFeatureMap provides an advantage to ZZFeatureMap. However, the difference is a lot bigger for RealAmplitudes, TwoLocal and EfficientSU2, than for PauliTwoDesign.



(a) Ansatz vs. Feature Map (b) Feature Map Entanglement vs. Optimizer

Figure 5.13: QML California with Noise: Correlations

Finally, Figure 5.13b compares feature map entanglement and optimizers. Interestingly, COBYLA works best when using no feature map entanglement (ZFeatureMap) and slightly better than the other optimizers when considering 'full' feature map entanglement. For the other strategies, SPSA seems to provide an advantage.

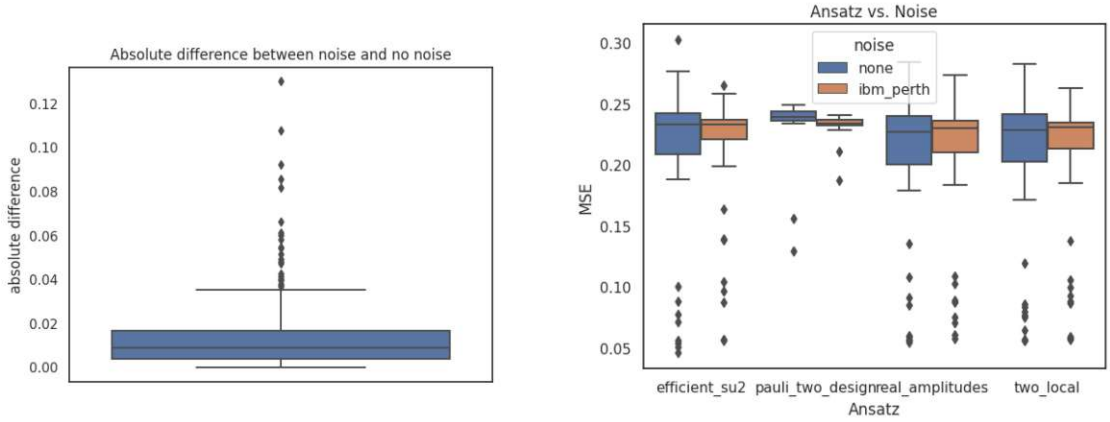
5.3.1.3 Comparison

Finally, we compare the results with and without noise. We find that the same configuration with and without noise has an average MSE difference of 0.015 and a standard deviation of 0.018. There are configurations that have almost no difference, but the maximum is at 0.13. Figure 5.14a shows how the differences are distributed. The differences are quite small, however, there are several outliers.

As mentioned before, we observe that noise actually increases the performance of the worst configurations, but also decreases the performance of the best ones. We show the behavior in Figure 5.14b, for every ansatz separately. Introducing noise in the circuit actually decreases the span of the box plots. The best and worst configurations are all no-noise ones.

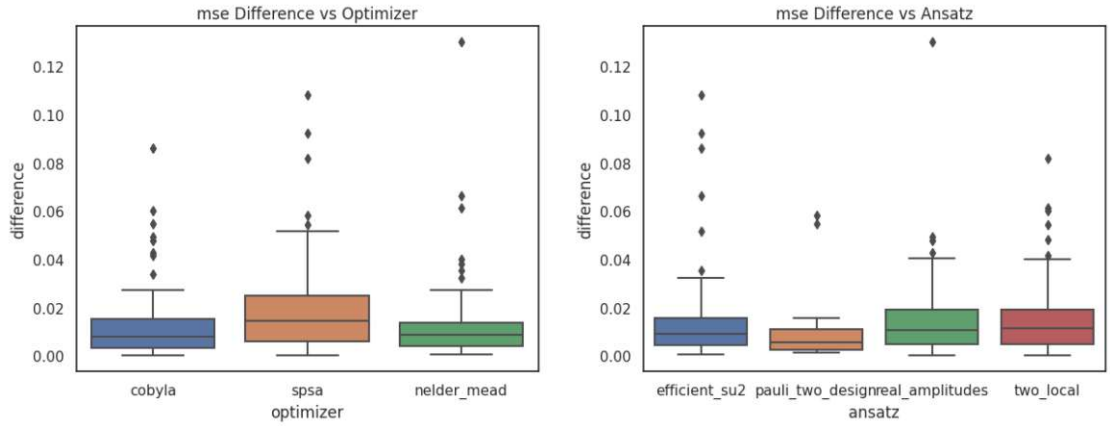
In the following, we compare each configuration with and without noise, i.e., we conduct a paired non-parametric Wilcoxon test. We extract the most and least similar configurations by taking those that are within a 10% span of the smallest and biggest difference.

Optimizer We find significant differences for SPSA, where the noisy experiments perform worse, and Nelder-Mead, where noisy experiments perform better. While we find all three optimizers in the set of most similar configurations, only one Nelder-Mead one makes up the set of least similar ones. SPSA is, however, slightly underrepresented among the most similar configurations with only 23% of the configurations being there,



(a) Absolute Difference in Accuracy (b) Ansatz vs. Noise

Figure 5.14: QML California: Absolute Differences and Ansatz

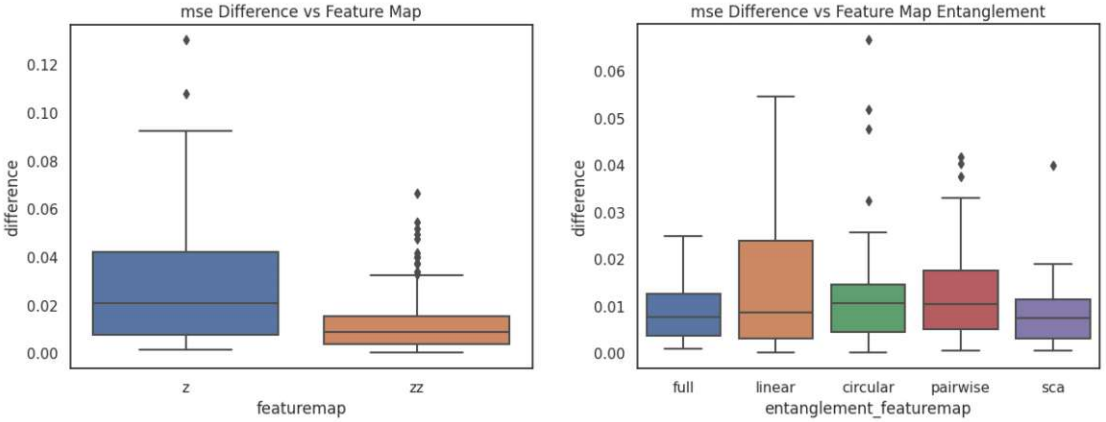


(a) Optimizer (b) Ansatz

Figure 5.15: QML California: Comparison of Optimizer and Ansatz

compared to 34% and 36% for COBYLA and Nelder-Mead, respectively. Figure 5.15a shows that the mean difference for COBYLA and Nelder-Mead is a lot smaller than for SPSA. Furthermore, except for some outliers, the Nelder-Mead configurations do not perform too differently with and without noise.

Ansatz We find significant performance differences for PauliTwoDesign and RealAmplitudes. Interestingly, the largest ratio of ansatzes among the set of most similar configurations is PauliTwoDesign (38%), with EfficientSU2 (34%) in second place. Only 30% and 29% of TwoLocal and PauliTwoDesign configurations make it into the set. Similarly, the set of least similar configurations consists of only one RealAmplitudes one.



(a) Feature Map (b) Feature Map Entanglement

Figure 5.16: QML California: Comparison of Feature Map and Entanglement

Ansatz Entanglement We do not find significant differences for any ansatz entanglement strategy. The least similar configuration uses 'sca' entanglement. 'Pairwise' and 'linear' entanglement are most often represented in the set of most similar configurations with 36% and 33% of configurations, respectively. Only 26% of 'full' entanglement configurations are in the set, compared to 31% for both 'sca' and 'circular'.

Feature Map We find significant differences for the ZFeatureMap, with noisy configurations having a 0.015 higher MSE on average. No such difference can be observed for ZZFeatureMap. Therefore, we find 34% of ZZFeatureMap configurations in the set of most similar ones, compared to only 19% for ZFeatureMap. The least similar configuration uses ZFeatureMap. Figure 5.16a visualizes the differences depending on the feature map. It can be seen that the differences are a lot higher for the ZFeatureMap than the ZZFeatureMap.

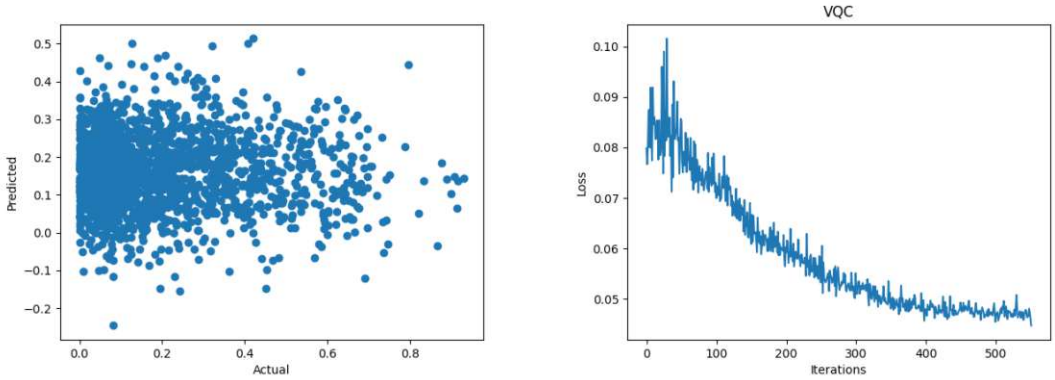
Feature Map Entanglement We find no significant differences for any entanglement strategy, except for 'none' (ZFeatureMap). 'full' and 'sca' entanglement configurations being slightly overrepresented with 39%, compared to only 28% and 29% for 'linear' and 'pairwise'. Figure 5.16b visualizes this pattern.

5.3.2 Seoul Bike Sharing Dataset

In the following, we will discuss our results for the Seoul Bike Sharing dataset. We will again first discuss the experiments without noise and afterward in the presence of noise.

5.3.2.1 Noiseless Configuration

We show the predicted vs. actual plot for the final results in Figure 5.17a. It can be seen that the model predicts only in range $[-0.2, 0.5]$. Unfortunately, the test performance is



(a) Predicted vs. Actual (b) Convergence

Figure 5.17: QML Seoul: Final Predictions

Ansatz	Optimizer	Feature Map	Entanglement	MSE	MAE	Time
EfficientSU2	SPSA	Z	circular	0.0209	0.1074	23993s
RealAmplitudes	SPSA	Z	full	0.0219	0.1113	27585s
TwoLocal	SPSA	Z	sca	0.0227	0.1172	22055s
EfficientSU2	SPSA	Z	full	0.0229	0.1161	31995s
TwoLocal	SPSA	Z	linear	0.0245	0.1209	21314s

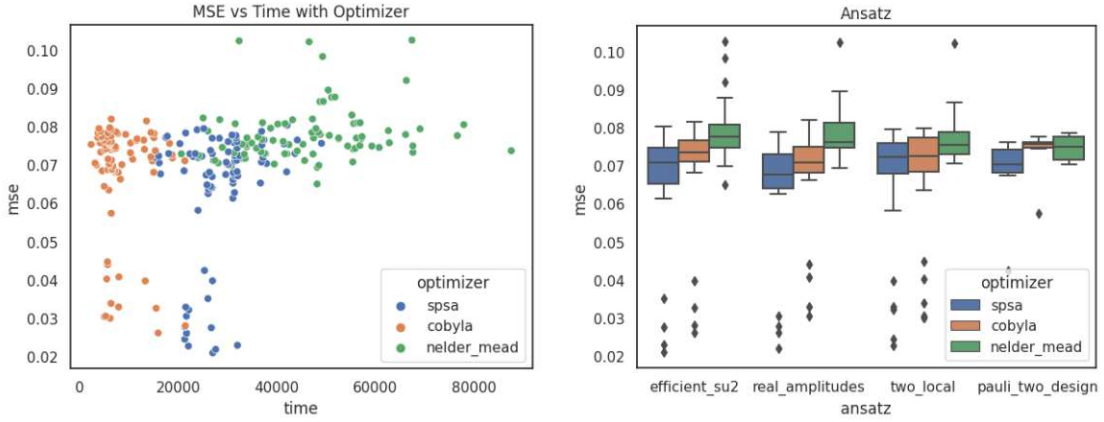
Table 5.5: QML Seoul: Top 5 Configurations

a lot worse than the best validation performance during hyperparameter tuning, with a MSE of 0.0467 and MAE of 0.1627. In particular, the model predicts only a very narrow range, leading to unsatisfying results. The convergence plot shows steady convergence, although at a high loss, hence, we hypothesize that it got stuck in a BP.

The results from the hyperparameter tuning show that the MSE values range from 0.02 to 0.10. The MAE ranges from 0.10-0.25. We show the top five models and parameters in Table 4.2. We will again analyze the results separately for every parameter below.

Optimizer We find that Nelder-Mead does not find a reasonable optimization path, similar to the California housing dataset. Therefore, both COBYLA and SPSA perform significantly better than Nelder-Mead. Furthermore, SPSA is significantly better than COBYLA, with the top five configurations being SPSA ones. Figure 5.18a shows the behavior as a scatter plot. SPSA configurations again take a lot longer to train than COBYLA ones do.

Ansatz We find no significant differences between the ansatzes. Nonetheless. PauliTwoDesign is not represented among the best-performing configurations, while five out of the best eight configurations use EfficientSU2. We show the performances in Figure 5.18b.



(a) Optimizer (b) Ansatz

Figure 5.18: QML Seoul: Optimizer and Ansatz

It can be seen that the RealAmplitudes and EfficientSU2 configurations tend to perform slightly better on average than the other ones, however, not when using Nelder-Mead.

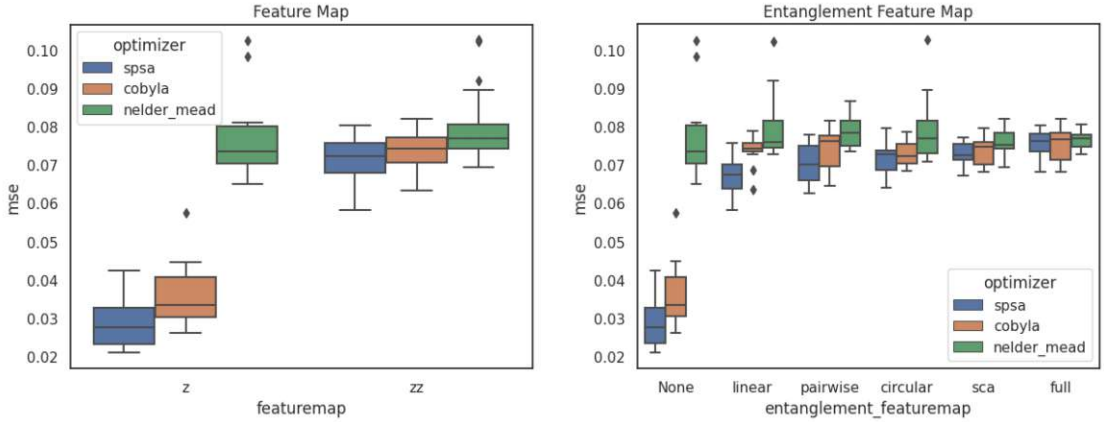
Ansatz Entanglement We find no significant differences regarding the ansatz entanglement neither overall, nor when testing in combination with the respective ansatz.

Feature Map ZFeatureMap performs significantly better than ZZFeatureMap, and we find that the best 27 configurations all use ZFeatureMap. Interestingly, we also observe a substantially higher standard deviation of the MSE for ZFeatureMap configurations (0.02), than we do for ZZFeatureMap ones (0.005). Figure 5.19a shows the performance of the feature maps as box plots. It can be seen that the advantage of ZFeatureMap is only given if COBYLA or SPSA are used for optimization.

Feature Map Entanglement No entanglement (ZFeatureMap) significantly outperforms all other strategies. Among the strategies for the ZZFeatureMap we find no significant differences. Figure 5.19b visualizes the entanglement.

Only Nelder-Mead configurations make up the set of worst configurations. Interestingly, no PauliTwoDesign configurations are in the set, i.e. while they do not perform the best, they do not make up the set of worst configurations either. We find a larger percentage of ZFeatureMap (4%) than ZZFeatureMap (0.9%) configurations in the set, suggesting, that the performance range grows bigger, but not necessarily all configurations perform well. Finally, we want to point out that all ZZFeatureMap configurations in the set of worst configurations use 'full' feature map entanglement.

Moreover, we want to investigate interesting patterns we found among the different parameters. When we combine the ansatz and feature map, we find that ZFeatureMap outperforms ZZFeatureMap for all ansatzes, but the advantage is not independent of the



(a) Feature Map (b) Feature Map Entanglement

Figure 5.19: QML Seoul: Feature Map and Entanglement

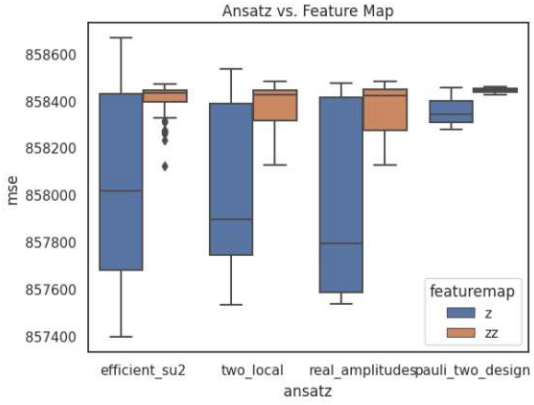
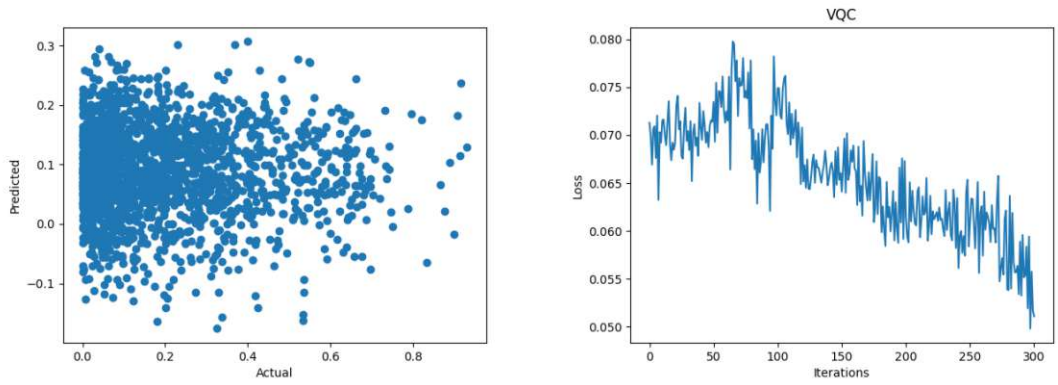


Figure 5.20: QML Seoul: Ansatz vs. Feature Map

ansatz. We plot the behavior in Figure 5.20. It can be seen that the advantage that ZFeatureMap can provide, is a lot smaller for the PauliTwoDesign ansatz than for the other ansatzes.

5.3.2.2 Noisy Configuration

We show the predicted vs. actual plot for the final results in Figure 5.21a and the convergence plot in Figure 5.17b. The predictions are similar to the noiseless experiments. The MSE is at 0.0547, hence, slightly better than without noise, and the MAE is at 0.1687. Both the noisy and noiseless models use SPSA as the optimizer. When comparing the convergence plots, we can observe that the noisy model's optimization path is a lot more volatile. While the final loss is very similar to the noiseless one, it can be seen that it has problems finding a reasonable path.



(a) Predicted vs. Actual (b) Convergence

Figure 5.21: QML Seoul with Noise: Final Predictions

Ansatz	Optimizer	Feature Map	Entanglement	MSE	MAE	Time
RealAmplitudes	SPSA	Z	circular	0.0218	0.1182	23993s
EffiienSU2	SPSA	Z	linear	0.0227	0.1169	27585s
TwoLocal	SPSA	Z	circular	0.0270	0.1245	22055s
EfficientSU2	SPSA	Z	circular	0.0279	0.1237	31995s
TwoLocal	SPSA	Z	sca	0.0245	0.1264	21314s

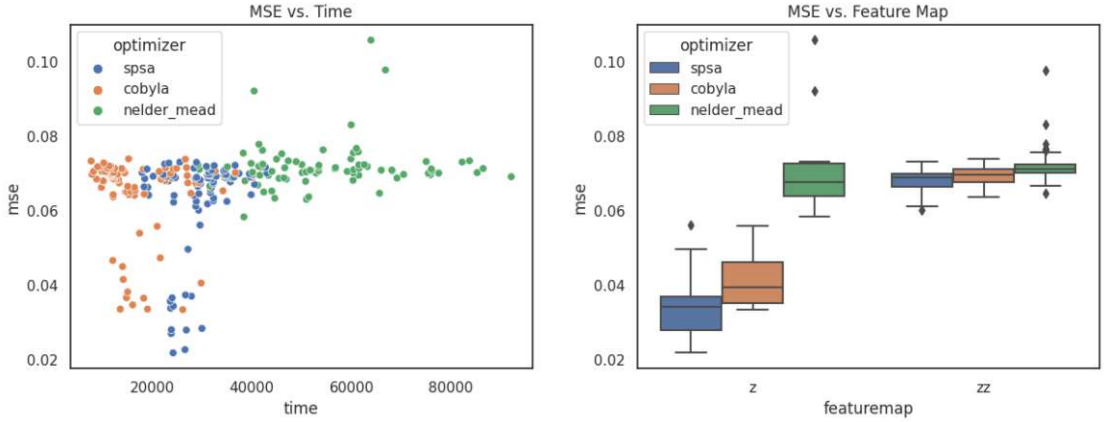
Table 5.6: QML Seoul with Noise: Top 5 Configurations

We give an overview of the best configurations with noise in Table 5.6. Again, all best configurations use the ZFeatureMap and SPSA as the optimizer. As before, we will elaborate on each hyperparameter separately in the following.

Optimizer Similarly to previous experiments, COBYLA and SPSA perform significantly better than Nelder-Mead, with a mean MSE of 0.064, 0.062 and 0.071 respectively. Furthermore, SPSA is significantly better than COBYLA. All best configurations use SPSA. We show a scatter plot, highlighting the different optimizers in Figure 5.22a.

Ansatz We find no significant differences between the different ansatzes. However, no PauliTwoDesign configurations are among the best-performing ones, and EfficientSU2 configurations are the most frequently used ones in the set (three times, compared to two TwoLocal and one RealAmplitudes).

Ansatz Entanglement We cannot observe significant differences between the different entanglement strategies. This holds even if we compare the entanglement strategies for every ansatz separately. However, 'circular' entanglement is used most frequently in the set of best configurations (three times, compared to one for 'full', 'linear' and 'sca').



(a) Optimizer (b) Feature Map

Figure 5.22: QML Seoul with Noise: Optimizer and Feature Map

Feature Map As previously observed, ZFeatureMap is significantly better than ZZFeatureMap with a mean MSE of 0.0489 compared to 0.0697. All best configurations use ZFeatureMap. The feature maps with respect to the optimizer is plotted in Figure 5.22b. It can be seen that the span of ZFeatureMap configurations is higher than the ones for ZZFeatureMap, but the Nelder-Mead configurations still perform similarly in mean for both feature maps.

Feature Map Entanglement Besides no entanglement (ZFeatureMap) being significantly better than any other entanglement strategy, we find no significant differences between the strategies.

The set of worst configurations is made up of only Nelder-Mead configurations, using the EfficientSU2 ansatz with full entanglement. Interestingly, the ratio of ZFeatureMap configurations (2%) is bigger than the ZZFeatureMap one (0.4%).

Finally, we can again observe the interplay between ansatz and feature map as previously reported, showing that the advantage of ZFeatureMap over ZZFeatureMap is dependent on the ansatz used.

5.3.2.3 Comparison

Finally, we want to directly compare the configurations with and without noise. The absolute differences between the configurations are quite small, as can be seen in Figure 5.23. The mean difference is at 0.6% and the standard deviation at 0.5%. The maximum we observe lies at 3%.

While we again find that the best configurations are most impacted by noise in the quantum computer, we see that noise can improve the mean performance.

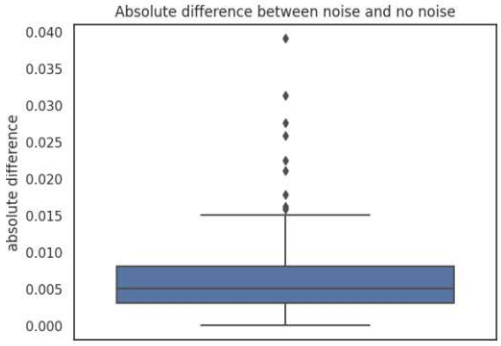


Figure 5.23: QML Seoul: Absolute Differences

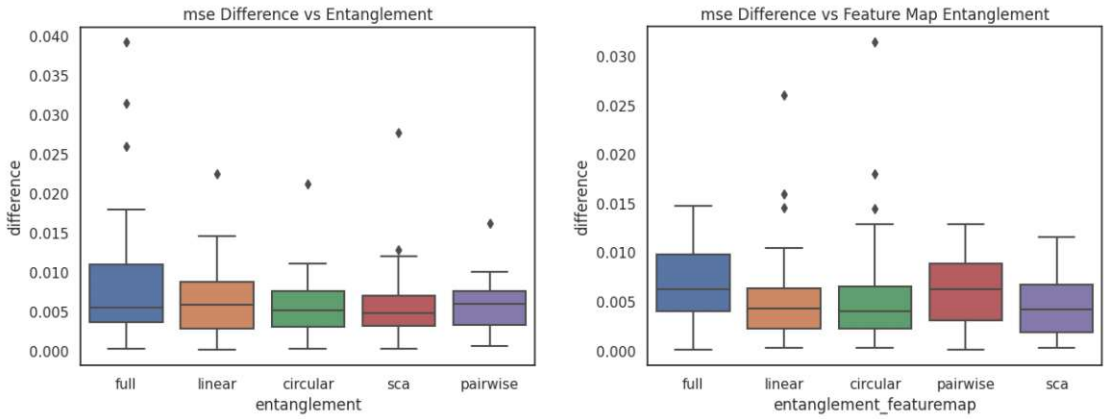
Optimizer We find significant differences in performance for all optimizers, with the noisy configurations performing slightly better in mean. The difference is not too big, with 0.3%, 0.7% and 0.2% mean absolute difference in accuracy for COBYLA, Nelder-Mead and SPSA respectively.

Ansatz Similarly to the optimizers, we find significant differences with and without noise for all ansatzes, with the noisy configurations performing slightly better in mean (between 0.2%-0.4%). Interestingly, we find a higher ratio of PauliTwoDesign configurations (25%) than for the other ones (18%-19%) in the set of most similar configurations.

Ansatz Entanglement There are significant differences in entanglement as well. This holds if we consider the entanglement strategies alone, as well as in combination with the respective ansatz. For the latter, there are significant differences for all but the EfficientSU2 ansatz with 'sca' entanglement. We plot the strategies in Figure 5.24a. 'Full' entanglement exhibits some outliers and shows a bigger IQR than the other strategies do. Nonetheless, 'pairwise' performs worst in mean, which is also represented in the set of most similar configurations, where only 13% of 'pairwise' configurations can be found, as compared to 16%-21% for the other strategies.

Feature Map We find significant differences for the ZZFeatureMap, with the noisy configurations achieving 0.5% more accuracy in mean. We cannot report significant differences for ZFeatureMap, however. Interestingly, we still find 20% of ZZFeatureMap configurations in the set of most similar ones, compared to 13% of ZFeatureMap ones.

Feature Map Entanglement Similarly, all but 'none' feature map entanglement (ZFeatureMap) show significant differences with and without noise. We show box plots with absolute differences for every entanglement strategy in Figure 5.24b. It can be seen that 'linear', 'sca' and 'circular' seem to lead to smaller differences, except for some outliers.



(a) Ansatz Entanglement (b) Feature Map Entanglement

Figure 5.24: QML Seoul: Absolute Differences Entanglement

5.3.3 Cover Type Dataset

In the following, we will present the results for the cover type dataset first on a perfect machine, then with noise.

5.3.3.1 Noiseless Configuration

The best-performing model during hyperparameter tuning uses COBYLA, the ZFeatureMap, the TwoLocal ansatz with pairwise entanglement and LDA for feature reduction. After retraining the model using 5,000 samples, we obtain an accuracy of 0.596 and an f-1 score of 0.581. It takes about 14 hours to train.

We show the confusion matrix in Figure 5.25. It can be seen that the model primarily predicts classes `spruce/fir` and `lodgepole pine`, which are the ones that are most common in the dataset. It manages to predict 23% of the `krummholz` samples correctly and about 16% of the `cottonwood/willow` ones. The results show that the model fails to generalize.

The convergence graph reveals that, while the process is quite smooth, it converges with a relatively high loss. The final training loss is about 1.8, as can be seen in Figure 5.26, which is still high. As a reference, some of the best configurations during tuning converged at a loss of 1.6.

We will now discuss the different configurations during tuning. Table 5.7 shows the ansatz, optimizer, feature map, ansatz entanglement, preprocessing, validation accuracy, validation f-1 score and training time for the best five configurations. Furthermore, in Appendix D in Table D.5, we list the top 40 configurations. The validation accuracies range from 8%-58%, i.e. there are significant performance differences between the configurations.

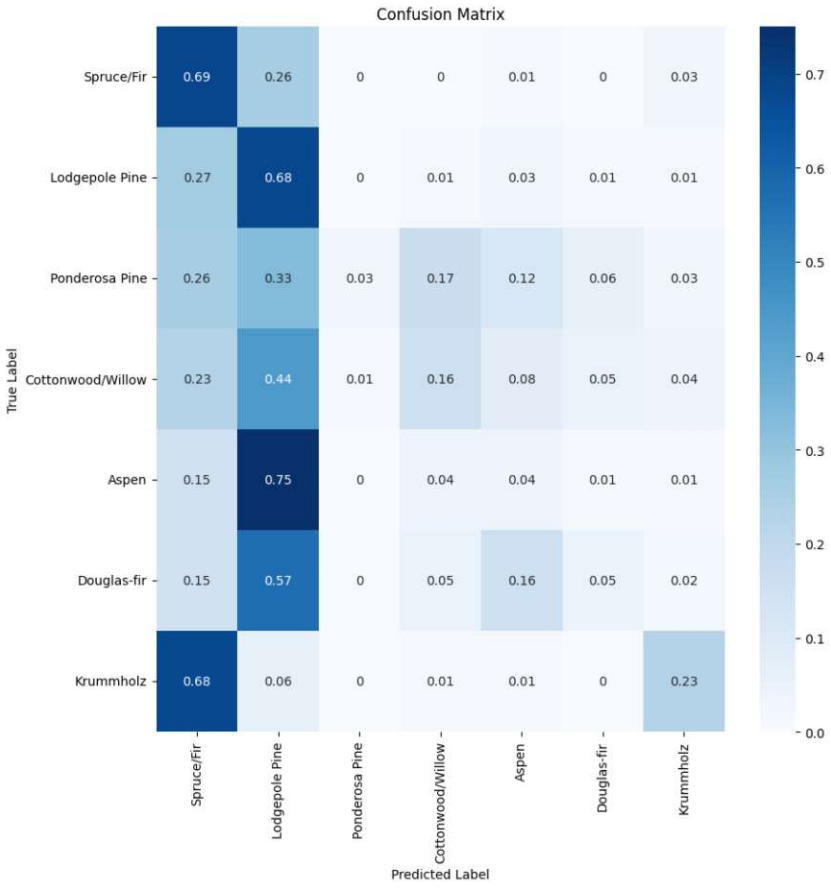


Figure 5.25: QML Cover Type: Confusion Matrix

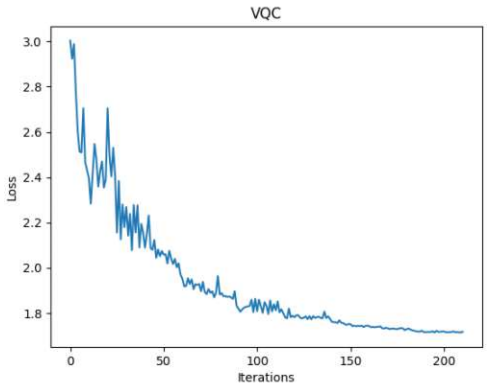


Figure 5.26: QML Cover Type: Convergence

Ansatz	Optimizer	Feature Map	Entang.	Prepr.	Acc.	F-1	Time
TwoLocal	COBYLA	Z	pairwise	LDA	0.588	0.583	606s
EfficientSU2	SPSA	Z	sca	LDA	0.576	0.576	3171s
EfficientSU2	SPSA	Z	circular	LDA	0.572	0.569	3208s
TwoLocal	COBYLA	Z	circular	LDA	0.584	0.564	1035s
EfficientSU2	SPSA	Z	linear	LDA	0.588	0.563	3158s

Table 5.7: QML Cover Type: Top 5 Configurations

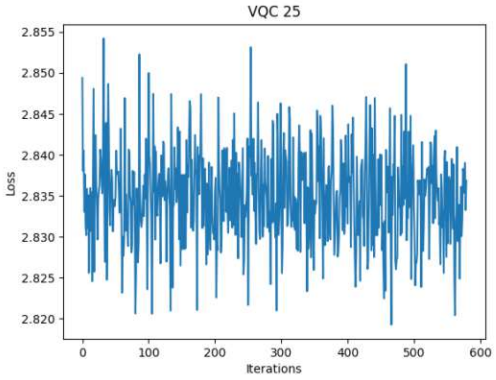
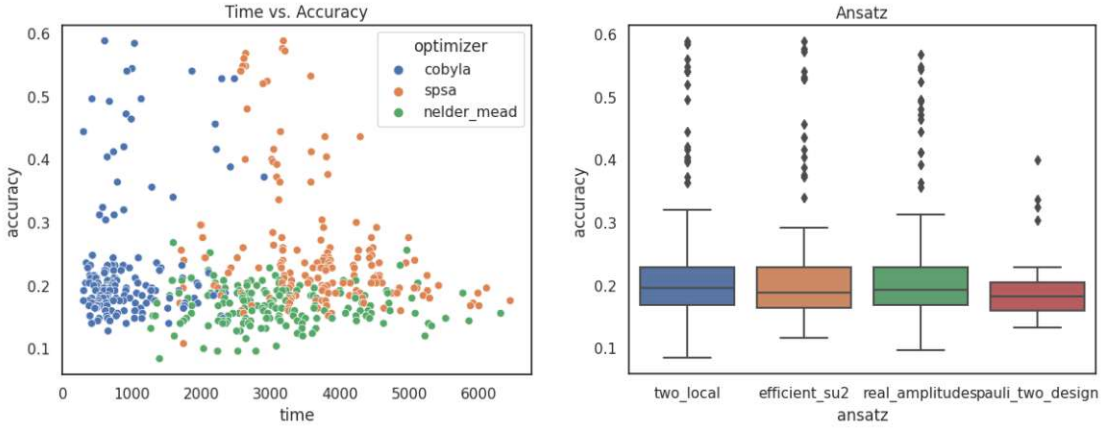


Figure 5.27: QML Cover Type: Nelder-Mead Convergence

Optimizer We find that both COBYLA and SPSA significantly outperform Nelder-Mead, which has a mean accuracy of only 16% (compared to 22% and 25%, respectively). Furthermore, SPSA significantly outperforms COBYLA. We visualize the performance, highlighting different optimizers in Figure 5.28a. It can be seen that the average SPSA and COBYLA configuration has an accuracy between 10%-30% and is slightly better than Nelder-Mead ones. Furthermore, while there are several SPSA and COBYLA configurations that outperform all others, there exists no such Nelder-Mead configuration. We find that Nelder-Mead fails to find a reasonable optimization path. Figure 5.27 shows an example of a convergence plot.

Ansatz There are no significant differences between the ansatzes. Nonetheless, we can observe that there are no PauliTwoDesign configurations among the best performing ones (the best is in place 37, with an accuracy of 40%). Figure 5.28b shows the performances of the different ansatzes as box plots. It can be seen that they perform similarly on average. While there are only few positive outliers for the PauliTwoDesign ansatz, there are several for the other three ansatzes.

Ansatz Entanglement Looking at the entanglement strategy for the ansatz, we cannot observe any significant differences. The same holds true when we consider combinations of ansatz and entanglement.



(a) Optimizer (b) Ansatz

Figure 5.28: QML Cover Type: Optimizer and Ansatz

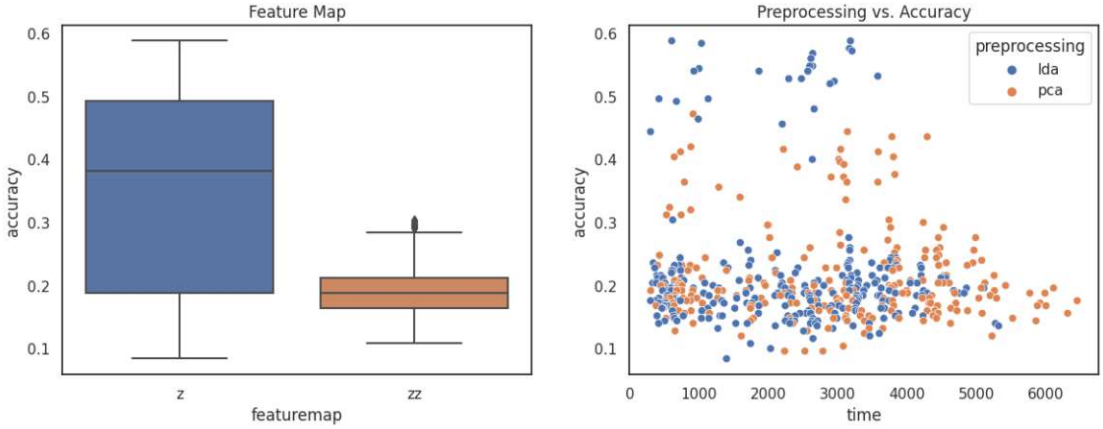
Feature Map ZFeatureMap significantly outperforms ZZFeatureMap in our experiments, with a mean accuracy of 35% and 19% respectively. The best 56 configurations all use ZFeatureMap. Figure 5.29a visualizes the feature map performances. The performance range of ZFeatureMap is a lot bigger than the one of ZZFeatureMap, suggesting that the performance of a configuration does not depend solely on the feature map, but rather multiple factors.

Feature Map Entanglement When considering the entanglement strategy for the ZZFeatureMap, we see that 'linear' entanglement outperforms all other strategies with a mean accuracy of 20%. Moreover, 'pairwise' (19%) works significantly better than 'full' (17%) and 'circular' (18%), and 'sca' (18%) works significantly better than 'full'.

Preprocessing We find that LDA (mean accuracy 22%) and PCA (21%) show no significant differences. Still, when considering the scatter plot in Figure 5.29b, it can be seen that all top-performing configurations use LDA as a preprocessing step. The techniques perform similarly in mean, but the best LDA configurations perform better than the best PCA configurations.

Furthermore, we analyze the set of worst configurations, which make up 40% of all configurations. We find that 66% of Nelder-Mead configurations are in the set, 36% of COBYLA and only 17% of the SPSA ones are. Moreover, 50% of the configurations using the PauliTwoDesign ansatz are among the worst-performing ones, with only 37%-40% for the other three ansatzes. 45% of the 'linear' ansatz entanglement configurations are in the set. The smallest fraction of ansatz entanglements among the worst-performing configurations is 'pairwise' entanglement, with a ratio of 33%.

Unsurprisingly, there are a lot less ZFeatureMap configurations (22%) than ZZFeatureMap (43%) in the set. When looking at the feature map entanglement, 54% of 'circular'



(a) Feature Map (b) Preprocessing

Figure 5.29: QML Cover Type: Feature Map and Preprocessing

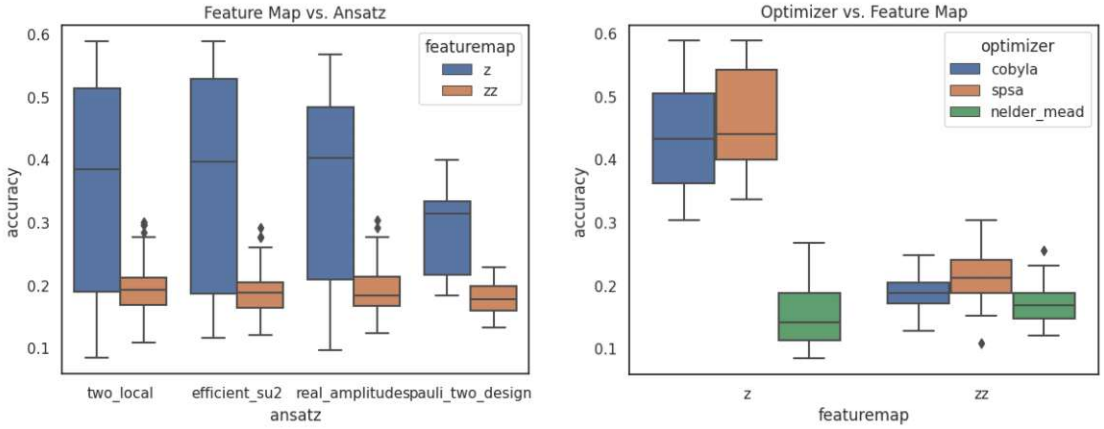
entanglement configurations can be found in the set. The smallest percentage is 'linear' entanglement with 30%, which is interesting, given that the only difference between the two is that the last qubit is not entangled with the first. Finally, slightly more LDA configurations (41%) than PCA configurations (38%) are among the worst-performing ones.

Lastly, we analyze correlations between different parameters. In particular, Figure 5.30a visualizes the performance of different ansatzes, highlighting the feature map applied. It can be seen that the ZZFeatureMap configurations perform similarly across all ansatzes. The performance range increases for the ZFeatureMap, i.e., one can find worse and better configurations than with the ZZFeatureMap. The performance when using the ZFeatureMap does, however, seem to be dependent on the ansatz used, especially the PauliTwoDesign ansatz performs worse than the others. It is still worth mentioning that, while no particularly good configurations are achieved, the performance range for the ansatz is quite narrow, and they do not make up the worst configurations overall either. The configurations using the ansatz therefore have a very stable performance.

Furthermore, we plot the performance of the feature map and optimizer combinations in Figure 5.30b. In particular, we want to point out that ZFeatureMap does not as a general rule outperform the ZZFeatureMap. The performance depends a lot on the optimizer, and an advantage is only achieved if the optimizer manages to find a reasonable optimization path. The ZFeatureMap with Nelder-Mead performs on average worse than it does with ZZFeatureMap and makes up for the worst configurations overall.

5.3.3.2 Noisy Configuration

The best model obtained during parameter tuning uses COBYLA, the ZFeatureMap, the RealAmplitudes ansatz with circular entanglement and LDA for feature reduction. We



(a) Ansatz vs. Feature Map

(b) Feature Map vs. Optimizer

Figure 5.30: QML Cover Type: Feature Map/Ansatz vs. Optimizer

Ansatz	Optimizer	Feature Map	Entang.	Prepr.	Acc.	F-1	Time
RealAmplitudes	COBYLA	Z	circular	LDA	0.560	0.550	2230s
TwoLocal	COBYLA	Z	pairwise	LDA	0.564	0.549	1730s
RealAmplitudes	SPSA	Z	linear	LDA	0.544	0.536	3016s
RealAmplitudes	SPSA	Z	circular	LDA	0.528	0.532	3043s
TwoLocal	COBYLA	Z	sca	LDA	0.532	0.528	2536s

Table 5.8: QML Cover Type with Noise: Top 5 Configurations

obtain an accuracy of 0.583, an f-1 score of 0.558, and the model trains for about 33 hours.

The confusion matrix is shown in Figure 5.31. We essentially see the same behavior as we did without noise, with the model mainly predicting the majority classes. The third highest per-class score is obtained on class `ponderosa pine`, where it only predicts the correct class in about 6.9% of the cases.

We will now again focus on the parameter tuning part. Table 5.8 shows the ansatz, optimizer, feature map, ansatz entanglement, preprocessing, validation accuracy, validation f-1 score and training time for the best five configurations when noise is introduced in the simulator. Furthermore, in Appendix D in Table D.6, we list the top 40 configurations. The validation accuracies range from 18%-56%. Interestingly, while the performances of best models decrease when noise is when noise is introduced, the mean accuracy again increases.

In the following, we will elaborate on the performance depending on the different parameters separately.

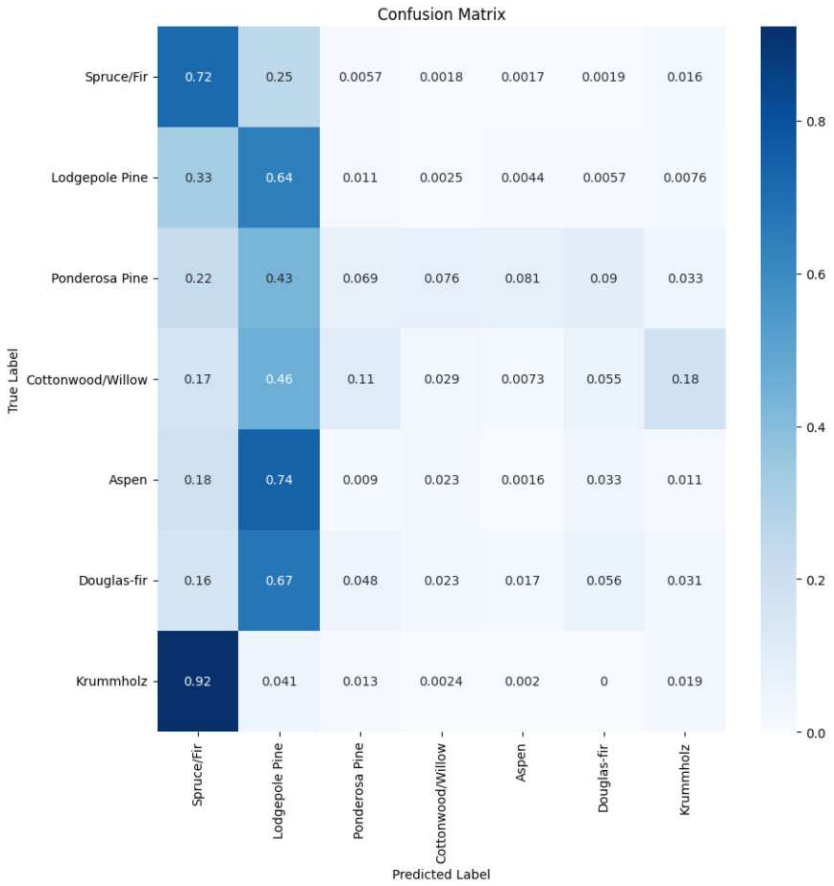
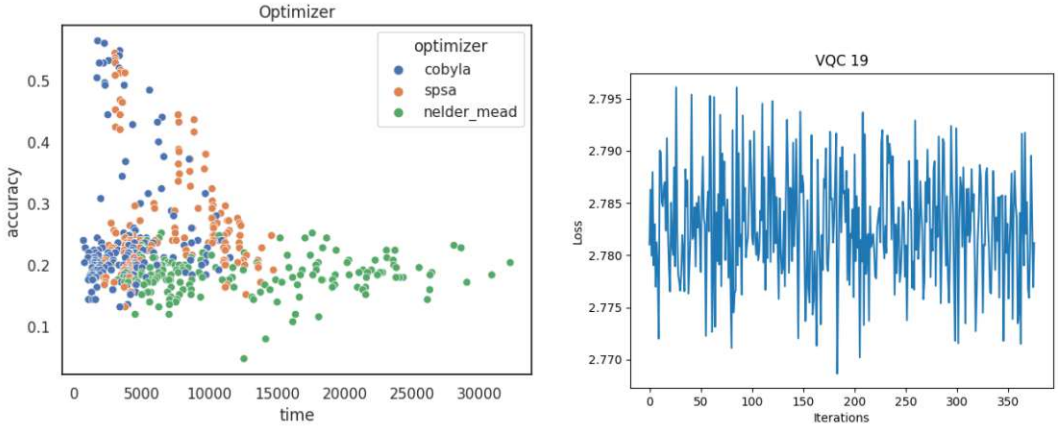


Figure 5.31: QML Cover Type: Confusion Matrix

Optimizer We find that both SPSA (mean accuracy 26%) and COBYLA (24%) outperform Nelder-Mead (18%). Additionally, SPSA is significantly better than COBYLA. While SPSA statistically outperforms COBYLA, we find that more configurations using COBYLA can be found among the top performing configurations (13 for COBYLA, compared to 10 for SPSA). Figure 5.32a visualizes the performances of the different optimizers. Moreover, Figure 5.32b again plots an example of the convergence of a Nelder-Mead configuration, showing that it fails to find a reasonable optimization path.

Ansatz We again find no significant differences regarding the chosen ansatz. However, PauliTwoDesign is underrepresented among the best configurations with only one candidate, as opposed to seven or eight for the other ansatzes.

Ansatz Entanglement There are no significant differences regarding the choice of entanglement. However, 'circular' and 'linear' entanglement can be found more often within the top configurations, occurring six and seven times, compared to one to four



(a) Optimizer (b) Nelder-Mead Convergence

Figure 5.32: QML Cover Type with Noise: Optimizer

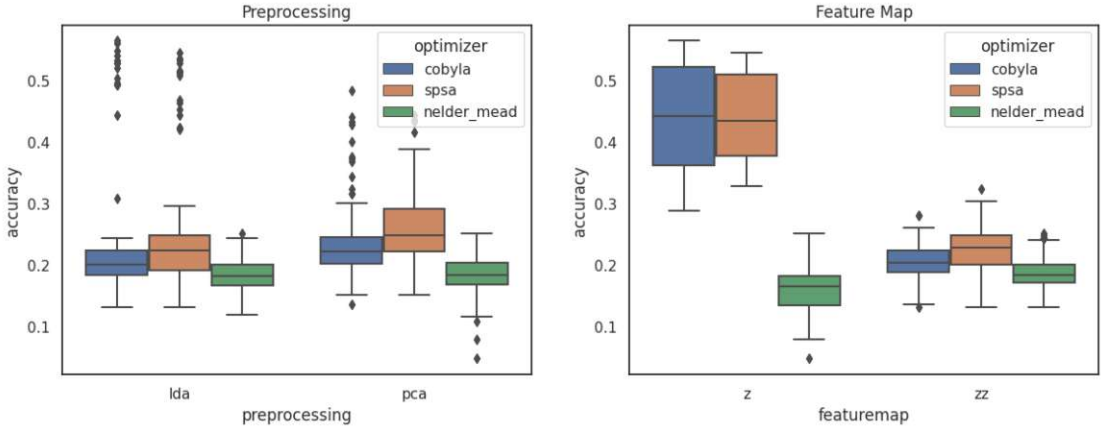
times for the others. When we consider the entanglement in combination with the ansatz, we find that for the EfficientSU2 ansatz, 'circular' (mean accuracy 23%) and 'full' (23%) outperform 'linear' (22%) and 'sca' (22%). Nonetheless, they are equally represented among the top-performing configurations.

Feature Map ZFeatureMap outperforms ZZFeatureMap with a mean accuracy of 34%, as compared to 20%. All the best-performing configurations use ZFeatureMap. We show box plots in Figure 5.33b, suggesting that using ZFeatureMap does not automatically lead to a better result. The performance still depends a lot on the optimizer used.

Feature Map Entanglement No entanglement (ZFeatureMap) outperforms all other strategies. There are no significant differences among the strategies otherwise.

Preprocessing PCA significantly outperforms LDA, with a mean accuracy of 22.9% compared to 22.8%. While the mean does not differ too much, the standard deviation is at 6% compared to 10% respectively, explaining the results of the significance tests. We visualize the performance of the different preprocessing techniques, depending on the optimizer, in Figure 5.33a. It can be seen that, while PCA performs significantly better, the best performing configurations use LDA.

Furthermore, we take a look at the set of worst configurations. We find that 12% of all Nelder-Mead configurations are in the set, compared to 3% and 0.5% of COBYLA and SPSA configurations. Interestingly, 11% of the ZFeatureMap configurations are among the worst performing ones, while it is only 4% of the ZZFeatureMap ones. Moreover, while only 1% and 2% of 'full' and 'pairwise' ansatz entanglement configurations are in the set, the ratio for 'linear' is 9%.



(a) Preprocessing (b) Feature Map

Figure 5.33: QML Cover Type with Noise: Preprocessing and Feature Map

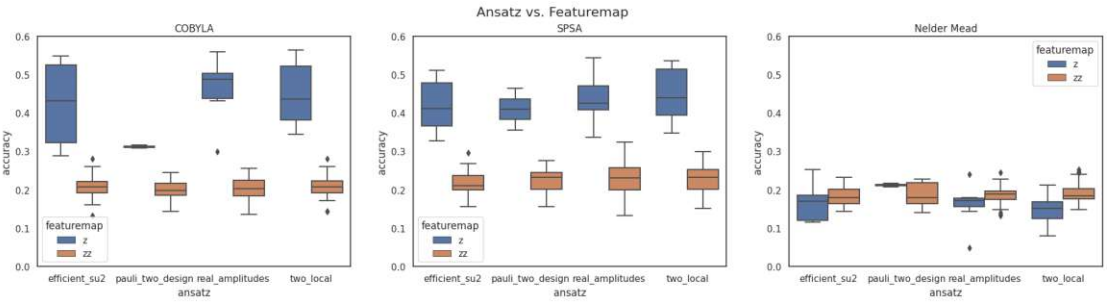


Figure 5.34: QML Cover Type with Noise: Ansatz vs. Feature Map

Finally, we plot the performance depending on the ansatz and feature map, separately for each optimizer, in Figure 5.34. It can be seen ZFeatureMap has a worse performance than ZZFeatureMap for many Nelder-Mead configurations. Furthermore, the performance range of the feature map depends on the ansatz. It can be seen that EfficientSU2 and TwoLocal tend to have a bigger range for COBYLA and SPSA, whereas PauliTwoDesign and RealAmplitudes have a more narrow one.

5.3.3.3 Comparison

Finally, we compare the results with and without noise. We find that the same configuration with and without noise has an average difference in accuracy of 3% and a standard deviation of 2%. There are configurations that have no difference, but the maximum is at 17%. Figure 5.35 shows how the differences are distributed. There are only few outliers.

We will now delve into the different parameters again and report on significant differences.

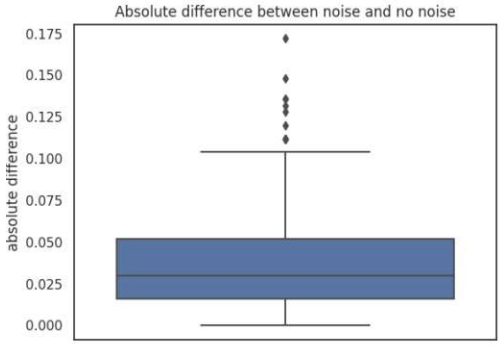


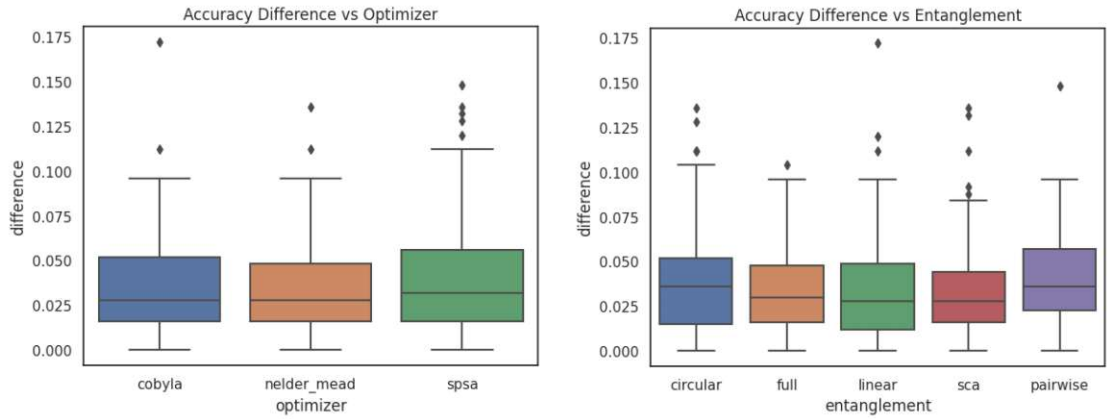
Figure 5.35: QML Cover Type: Absolute Difference in Accuracy

Optimizer We find no significant differences for SPSA (mean accuracy of 25% without and 26% with noise) but significant ones for COBYLA (22% without and 24% with) and Nelder-Mead (16% without and 18% with). Still, SPSA configurations make up the majority of configurations with the biggest performance difference. Figure 5.36a shows, however, that the differences between the optimizers are minimal.

Ansatz We find significant performance differences for all ansatzes. The mean accuracies for the noisy experiments are usually 1%-2% bigger than without noise, however, the standard deviation for EfficientSU2, RealAmplitudes and TwoLocal are with 8%-9% bigger than the PauliTwoDesign ones with 5%-6%. None of the PauliTwoDesign configurations are among the least similar configurations, while 0.5%-0.6% of the configurations using other ansatzes are in the set.

Ansatz Entanglement Significant differences can be found for 'circular', 'full' and 'linear' entanglement (with between 1%-2% mean difference). None of the 'full' entanglement configurations are among the least similar ones. Figure 5.36b visualizes the differences between the strategies. It can again be seen that the box plots for different entanglement strategies look very similarly, i.e., there are no clear differences between the different hyperparameter choices.

Feature Map We find significant differences for ZFeatureMap, where the noisy experiments perform worse (34% mean accuracy without compared to 25% with noise), but not for ZZFeatureMap (19% without and 20% with noise). 2% of the ZFeatureMap configurations are among the least similar configurations, whereas only 0.2% of ZZFeatureMap are in the set. Similarly, 38% of the ZZFeatureMap configurations are among the most similar ones, compared to only 32% for ZFeatureMap. Figure 5.37 visualizes the differences depending on the feature map.



(a) Optimizer (b) Entanglement

Figure 5.36: QML Cover Type: Comparison Optimizer and Entanglement

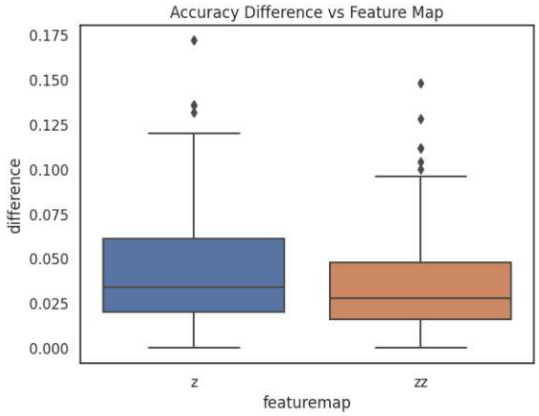


Figure 5.37: QML Cover Type Comparison: Feature Map

Feature Map Entanglement Significant differences can be found for 'circular', 'full', 'pairwise' and 'sca' entanglement with a 1%-3% mean difference in accuracy. No 'circular', 'full' or 'pairwise' feature map entanglement configurations can be found among the least similar configurations.

Preprocessing Both LDA and PCA perform significantly different with and without noise, with a 0.7% and 1.7% difference in mean accuracy and a slightly higher standard deviation with no noise.

5.3.4 KDD Cup 1999 Dataset

In the following, we will first discuss the results without noise, and then in presence of noise.

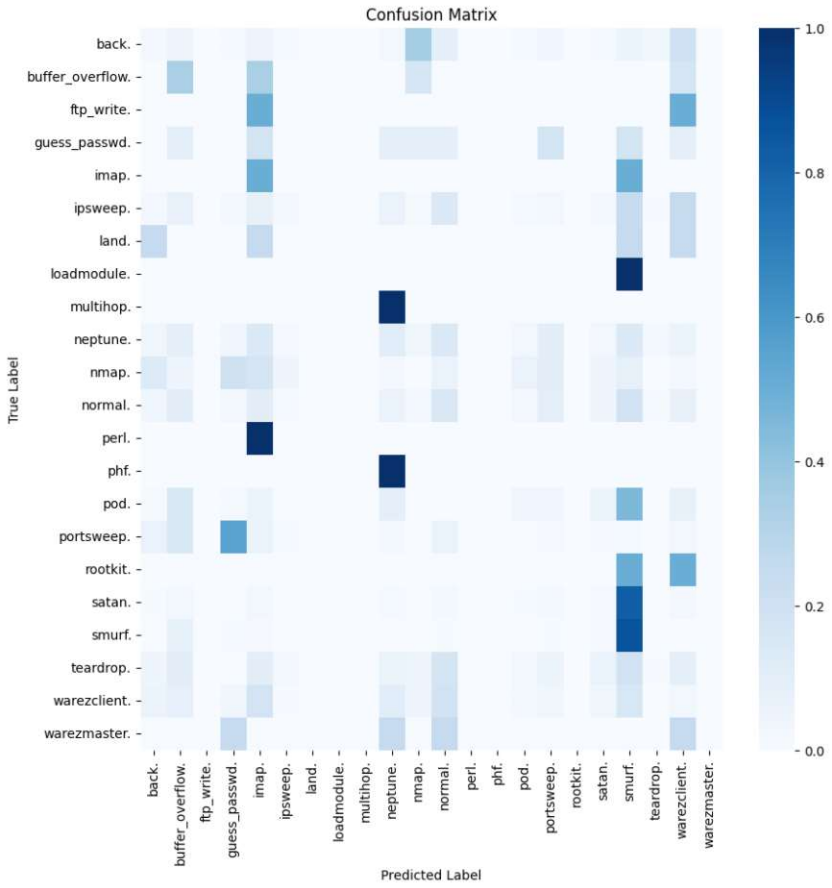


Figure 5.38: QML KDD: Confusion Matrix

5.3.4.1 Noiseless Configuration

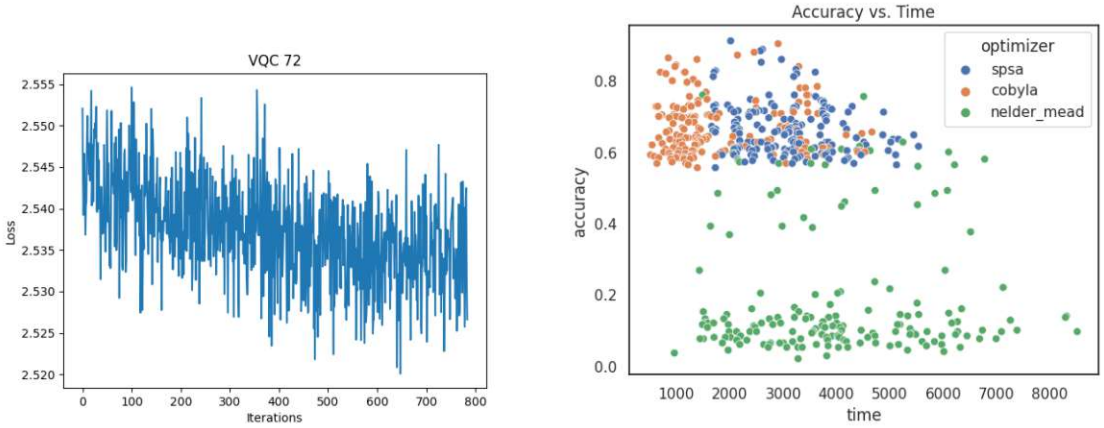
The loss for the final model stagnated quite early on at about 2.0, while it decreased to about 1.25 during the hyperparameter tuning phase. The final accuracy was at 54%, which is 37% lower than the best result obtained during hyperparameter tuning.

We show the confusion matrix in Figure 5.38, normalized by the true values. Some classes are hardly ever predicted (i.e. `ftp write`, `land`, `load module`, `multihop`, ...), while `smurf` and `neptune` attacks, which make up the classes in the dataset that are most often represented, are predicted frequently. Interestingly, it does not predict class `normal` too often, even though it is one of the majority classes in the dataset.

When looking at the tuning results, it can be seen that the validation performances range from 2% to 91% accuracy. All in all, we find that some configurations work very well, while certain hyperparameter configurations underperform altogether. Table 5.9 lists the ansatz, optimizer, feature map, entanglement, preprocessing, validation accuracy, validation f-1 score and time for the best five configurations. Furthermore, in Appendix

Ansatz	Optimizer	Feature Map	Entang.	Prepr.	Acc.	F-1	Time
TwoLocal	SPSA	Z	full	PCA	0.912	0.905	2016s
EfficientSU2	COBYLA	Z	circular	PCA	0.904	0.896	2907s
RealAmplitudes	SPSA	Z	sca	PCA	0.884	0.890	2576s
RealAmplitudes	SPSA	Z	circular	PCA	0.888	0.886	2612s
EfficientSU2	COBYLA	Z	sca	PCA	0.880	0.886	2457s

Table 5.9: QML KDD: Top 5 Configurations



(a) Convergence Nelder-Mead

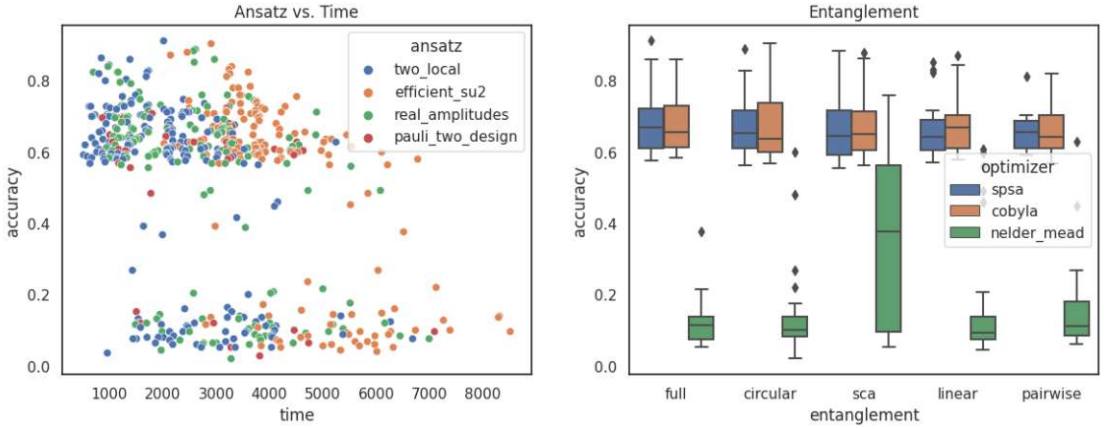
(b) Optimizer

Figure 5.39: QML KDD: Nelder-Mead Convergence and Optimizer

D in Table D.7, we list the top 40 configurations. We will discuss the performance for each parameter separately in the following.

Optimizer We find that both COBYLA and SPSA perform significantly better than Nelder-Mead. While both have a mean accuracy of about 66%, Nelder-Mead obtains only 18% in mean. We find the convergence plot for Nelder-Mead to be very unstable; an example can be seen in Figure 5.39a. There are no significant differences between SPSA and COBYLA. Figure 5.39b plots the time vs. accuracy, highlighting the different optimizers. It can be seen that there are few Nelder-Mead configurations with comparable performance to SPSA or COBYLA. Furthermore, SPSA takes, as a general rule, longer to fit than COBYLA.

Ansatz We find no significant differences between the performance of the different ansatzes, all of them achieving about 50% accuracy in mean. Still, the best PauliTwoDesign configuration is only in place 66 overall, with an accuracy of 72%. While it performs comparably to the other ansatzes in mean, there are no above-average configurations as can be observed for the other ansatzes. Figure 5.40a shows the time vs. accuracy, highlighting different ansatzes. While there is no clear correlation between ansatz and



(a) Ansatz (b) Entanglement

Figure 5.40: QML KDD: Ansatz and Entanglement

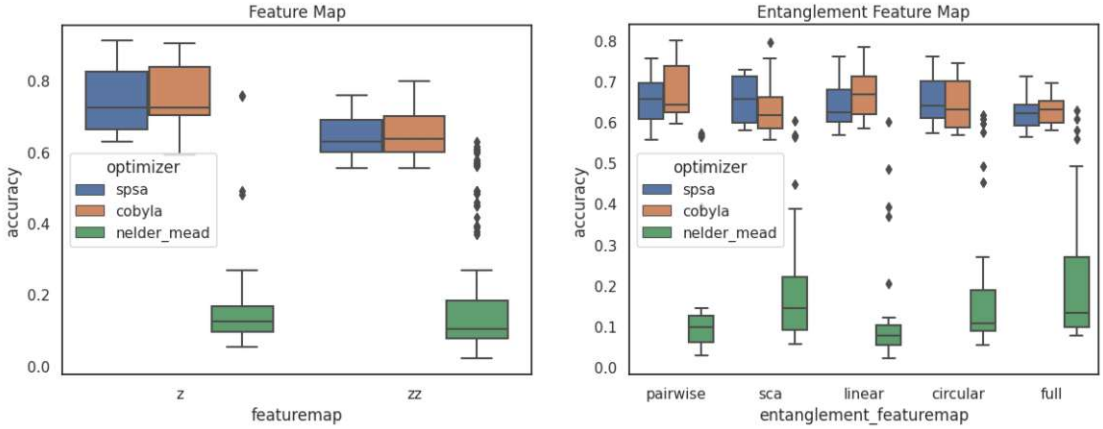
accuracy, it can be seen that EfficientSU2 usually takes longer to fit. Furthermore, as discussed previously, the PauliTwoDesign configurations lie mostly in the center.

Ansatz Entanglement There are no significant differences between the different entanglement strategies, not even in combination with the ansatz. Figure 5.40b compares different entanglement strategies. Entanglement 'none' corresponds to using the PauliTwoDesign ansatz. It can be seen that it performs well on average, but there are hardly any negative or positive outliers.

Feature Map We find that the ZFeatureMap significantly outperforms the ZZFeatureMap with a mean accuracy of 56% and 49% respectively and all best configurations use the ZFeatureMap. We compare the feature maps visually in Figure 5.41a. It can be seen that ZFeatureMap is not always better than ZZFeatureMap, rather the span of the performance is bigger. Furthermore, the two feature maps perform about the same when using Nelder-Mead as the optimizer.

Feature Map Entanglement No entanglement (ZFeatureMap) outperforms all other strategies significantly. Furthermore, we find 'circular' (mean accuracy of 50%) and 'pairwise' (49%) to outperform 'full' (48%), and 'pairwise' to outperform 'linear' (48%). A box plot is shown in Figure 5.41b. It can again be seen that the performance strongly depends on the optimizer.

Preprocessing Lastly, we compare the results of PCA and LDA. We find that PCA performs significantly better than LDA, with a mean accuracy of 53%, as compared to 47%. We visualize the different performances in Figure 5.42. It is interesting to see that the range of the LDA configurations is quite narrow, i.e., they mostly have about the same performance. For PCA, there are several positive outliers.



(a) Feature Map (b) Entanglement Feature Map

Figure 5.41: QML KDD: Feature Map and Entanglement Feature Map

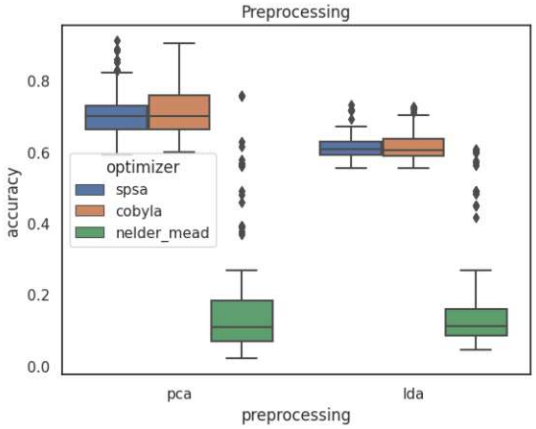
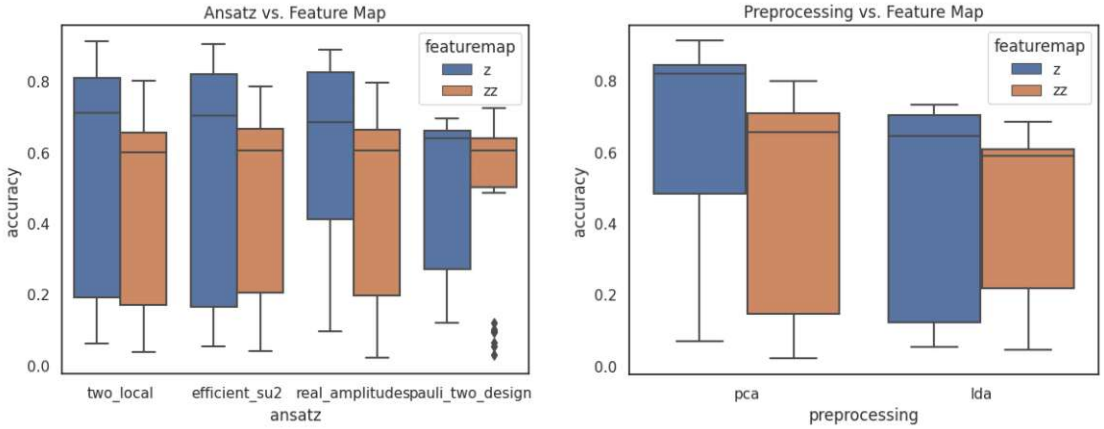


Figure 5.42: QML KDD: Preprocessing

Furthermore, we look at the worst configurations (within 10% of the worst accuracy). We find 58% of Nelder-Mead configurations in the set, and no configurations of other optimizers. Interestingly, only 12% of the configurations using 'sca' as entanglement strategy are within the worst configurations, compared to 20%-22% for the other entanglement strategies.

20% ZZFeatureMap configurations compared to 16% of the ZFeatureMap ones are within the set of worst configurations. We also find differences depending on the entanglement of the ZZFeatureMap. 27% and 25% of the 'linear' and 'pairwise' configurations, compared to 14% and 15% of the 'linear' and 'sca' configurations, are among the worst performing ones.

Finally, we look at correlations between different parameters. Figure 5.43a plots the performance of different ansatz and feature map combinations. Although ZFeatureMap



(a) Ansatz vs. Feature Map (b) Preprocessing vs. Feature Map

Figure 5.43: QML KDD: Ansatz/Preprocessing vs. Feature Map

significantly outperforms ZZFeatureMap, we find that the ansatz impacts the extent to which this holds true. It can be seen that the difference in performance of feature map is a lot smaller for the slightly underperforming PauliTwoDesign ansatz. The plot insinuates that ZFeatureMap does not, as a general rule, lead to better results, rather it amplifies the strengths of a good ansatz.

Furthermore, Figure 5.43b compares the feature map with the applied preprocessing. We observe a similar behavior with ZFeatureMap working better when using PCA as the preprocessing method.

5.3.4.2 Noisy Configuration

We observe a similar behavior as we do without noise, when looking at the results of the final model. We obtain an accuracy of 57% and again observe a big discrepancy (29%) between the obtained results during hyperparameter tuning and final training. The confusion matrix in Figure 5.44 shows that it mainly predicts the classes `smurf`, `buffer overflow` and `normal`.

When we analyze the hyperparameter tuning results, we observe that the accuracies range from 0.8%-86%. We list the top five configuration in Table 5.10. Furthermore, in Appendix D in Table D.8, we list the top 40 configurations.

Optimizer We find that the optimizers perform significantly different. In particular, both COBYLA (mean accuracy 66%) and SPSA (65%) outperform Nelder-Mead (15%). Additionally, we find that COBYLA is significantly better than SPSA. Still, they are equally represented among the best configurations. Figure 5.45a plots the time vs. accuracy, comparing the optimizers visually.

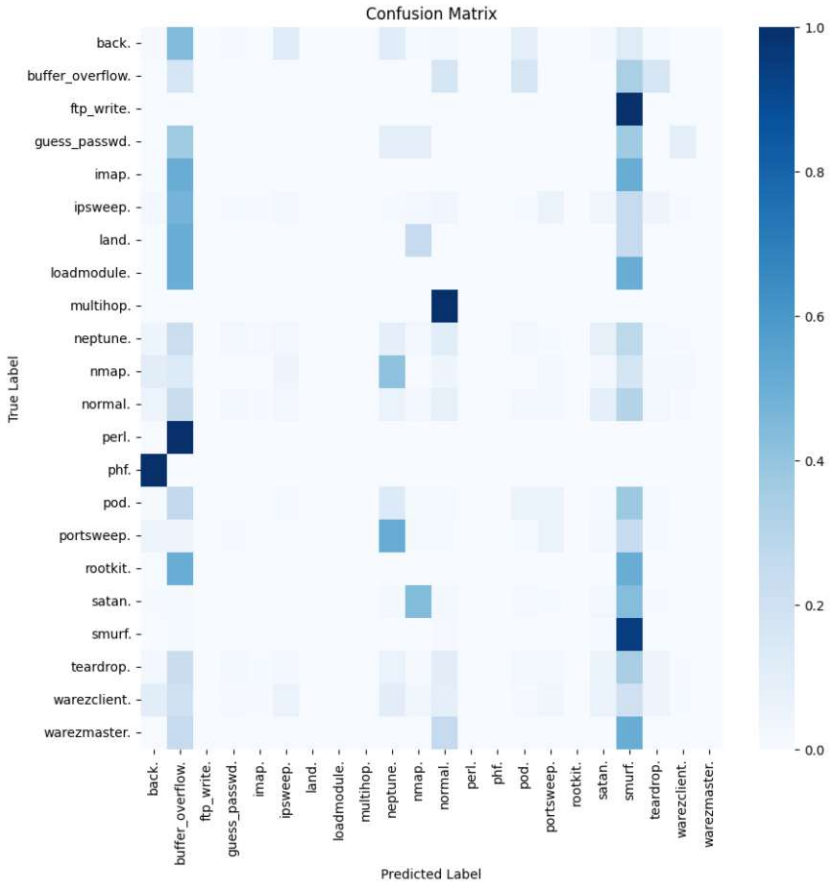
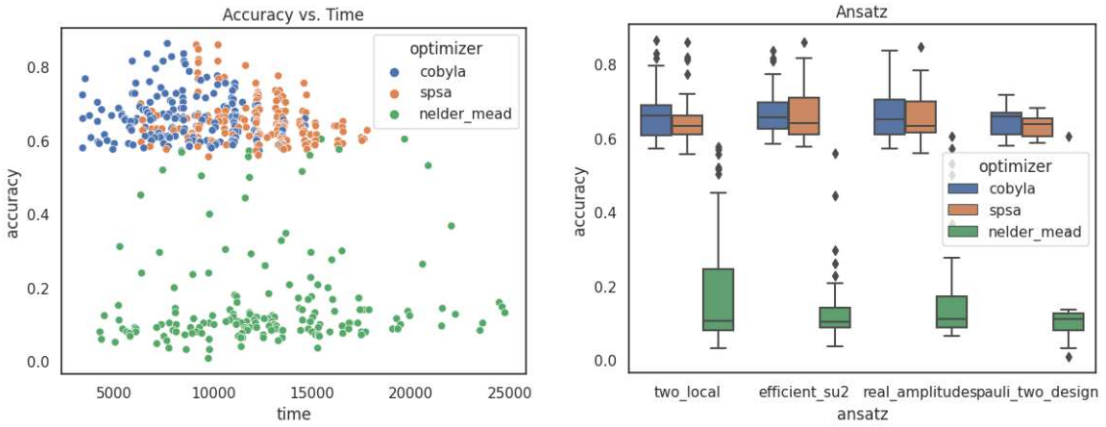


Figure 5.44: QML KDD with Noise: Confusion Matrix

Ansatz	Optimizer	Feature Map	Entang.	Prepr.	Acc.	F-1	Time
TwoLocal	COBYLA	Z	circular	PCA	0.864	0.859	7677s
EfficientSU2	SPSA	Z	circular	PCA	0.860	0.854	10241s
TwoLocal	SPSA	Z	linear	PCA	0.860	0.852	9142s
EfficientSU2	COBYLA	Z	sca	PCA	0.836	0.846	8504s
RealAmplitudes	SPSA	Z	circular	PCA	0.848	0.839	9245s

Table 5.10: QML KDD with Noise: Top 5 Configurations



(a) Optimizer (b) Ansatz

Figure 5.45: QML KDD with Noise: Optimizer and Ansatz

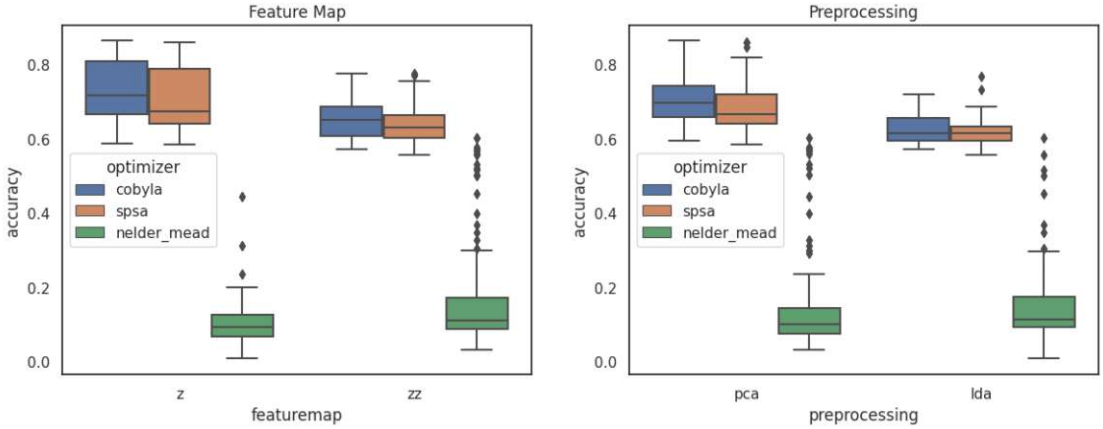
Ansatz Analyzing the ansatz, we find no significant differences between the different choices. However, it can again be seen that PauliTwoDesign is not represented at all among the best configurations. Figure 5.45b shows the performance of the ansatzes depending on the used optimizers. It can be seen that PauliTwoDesign does not in general perform worse. Rather, the performance range is quite narrow, without a lot of positive or negative outliers.

Ansatz Entanglement There are no significant differences between the entanglement strategies. This holds true overall, but also when comparing the entanglement strategies for every ansatz separately.

Feature Map Furthermore, we find that, similarly to the no-noise scenario, ZFeatureMap is significantly better than ZZFeatureMap, with an average accuracy of 51% and 48% respectively. Still, four configurations using the ZZFeatureMap made it into the best configurations. Figure 5.46a visualizes the results as box plots. It can be seen that the span of the performances of ZFeatureMap is larger than the ZZFeatureMap one, and that it is not, as a general rule, better. It is also interesting to see that there are a lot of positive outliers for Nelder-Mead configurations with ZZFeatureMap.

Feature Map Entanglement When considering the different entanglement strategies, we find no significant differences between them, except for 'no entanglement' (ZFeatureMap) outperforming all other configurations.

Preprocessing Finally, we consider the preprocessing applied. It can be seen that PCA works significantly better than LDA, with a mean accuracy of 51% compared to 46%. Figure 5.46b shows that the advantage is only given if a reasonable optimizer is employed.



(a) Feature Map (b) Preprocessing

Figure 5.46: QML KDD with Noise: Feature Map and Preprocessing

When looking at the worst configurations, we find a large fraction (52%) of Nelder-Mead configurations, while no configurations for SPSA and COBYLA made it into the set. 21% of the ZFeatureMap configurations are in the set, compared to 16% of the ZZFeatureMap ones. Moreover, 'sca' and 'linear' ansatz entanglements are slightly underrepresented, with 11% and 15% respectively, as compared to 19%-21% for the other strategies. Interestingly, only 4% of the 'full' feature map entanglement configurations are in the set, as compared to 15%-26% for the other entanglement strategies. The difference between PCA and LDA is less pronounced with 19% and 15% respectively.

Finally, we look at how different parameters affect each other. In particular, we consider how ansatz and feature map work with each other. Interestingly, we find significant differences between the Z- and ZZFeatureMap only for the EfficientSU2 and RealAmplitudes ansatz. The two feature maps do not show any significant differences when considering the PauliTwoDesign and TwoLocal ansatz. We plot the configurations, facetted by the optimizer used, in Figure 5.47. The plots for SPSA and COBYLA look similarly. Interestingly, the pattern seems to be reversed for Nelder-Mead, i.e., the ZZFeatureMap configurations appear to work better on average.

5.3.4.3 Comparison

Finally, we directly compare the configurations with and without noise and look for significant differences. We find that the same configurations have an average difference in accuracy of 6% and a standard deviation of 8%. There are configurations that have no difference, but the maximum is at 55%. Figure 5.48 shows how the differences are distributed. Although 75% of the configurations have a difference of less than 10%, there are a lot of outliers.

We will now delve into the different parameters again and report on significant differences.

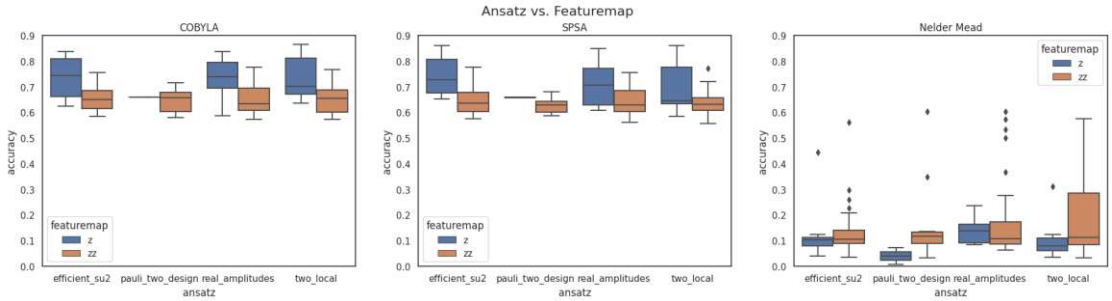


Figure 5.47: QML KDD with Noise: Ansatz vs. Feature Map

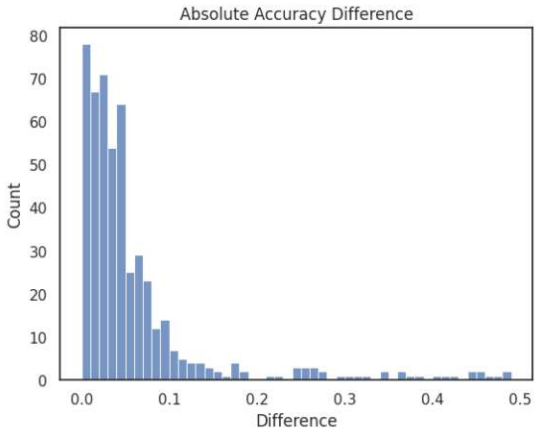
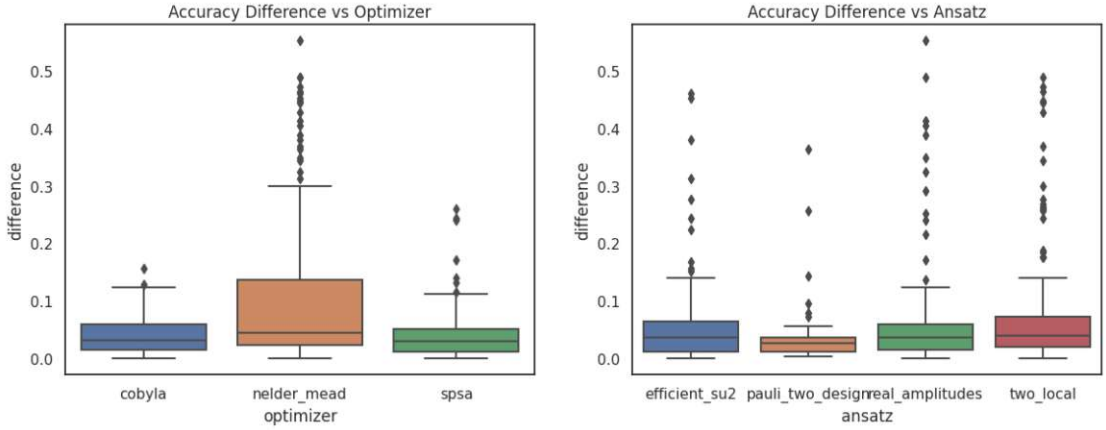


Figure 5.48: QML KDD: Absolute Difference in Accuracy

Optimizer We find that there are no significant differences for any optimizer. However, the least similar configurations consist of only Nelder-Mead configurations and while about 36% of COBYLA and SPSA configurations are within the most similar ones, only 27% of the Nelder-Mead ones are in the set. We visualize the results in Figure 5.49a.

Ansatz When considering the ansatz, we find significant performance differences for all except PauliTwoDesign. 40% of the PauliTwoDesign configurations are within the most similar ones, whereas only 30%-34% are in the most similar configurations from the other ansatzes. Interestingly, the least similar configurations are only made up from RealAmplitudes configurations. We show the accuracy differences of the ansatzes in Figure 5.49b.

Ansatz Entanglement We find no significant differences neither when considering the entanglement parameter alone, nor in combination with the ansatz. However, the least similar configurations are up of only 'sca' ones, hence, all of them use RealAmplitudes with 'sca' entanglement.



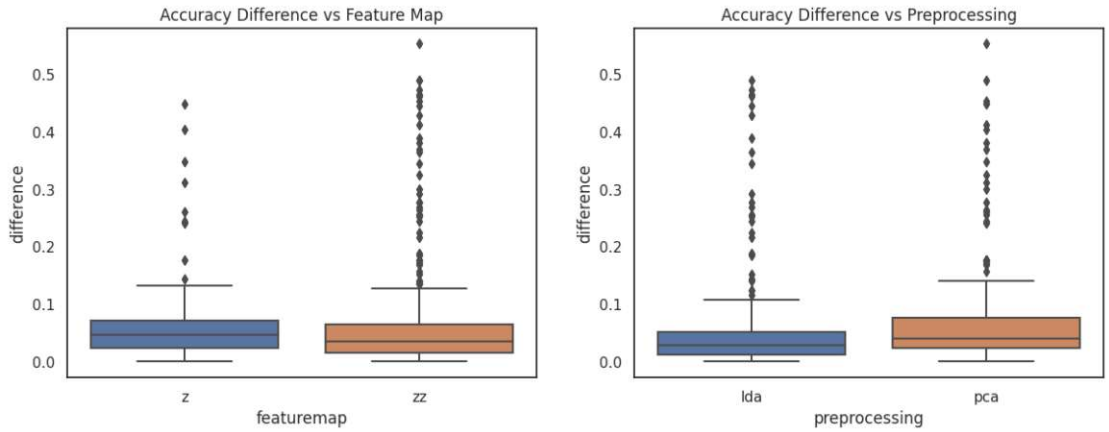
(a) Optimizer (b) Ansatz

Figure 5.49: QML KDD: Comparison Optimizer and Ansatz

Feature Map We find that the results for the ZFeatureMap are significantly different (mean accuracy of 51% with noise compared to 56% without). Interestingly, there are no significant differences for ZZFeatureMap (48% with and 49% without noise). 34% of the ZZFeatureMap configurations are within the most similar configurations, as compared to 28% for the ZFeatureMap. All configurations within the least similar ones use the ZZFeatureMap as well (0.1%). The results are interesting, as in the visualization in Figure 5.50a, there are a lot more outliers for ZZFeatureMap. Still, the minimum, 25, 50 and 75 quantile are lower for ZZFeatureMap than ZFeatureMap.

Feature Map Entanglement Similarly, we find that the performance for no feature map entanglement (ZFeatureMap) differs significantly, however, no differences can be found for the entanglement strategies of ZZFeatureMap. Interestingly, 38% of the 'full' feature map entanglement configurations are within the most similar configurations, compared to 31%-34% for the other strategies. Furthermore, all the least similar configurations use 'circular' entanglement.

Preprocessing We run the same analysis for the applied preprocessing and find no significant differences for LDA, but a significant difference for PCA (51% mean accuracy with noise and 53% without). All the least similar configurations use PCA. Furthermore, with 36% of the configurations, LDA is represented a lot more frequently among the set of most similar configurations than PCA with 29%. Figure 5.50b shows that the range for PCA is bigger than for LDA.



(a) Feature Map (b) Preprocessing

Figure 5.50: QML KDD Comparison: Feature Map and Preprocessing

5.4 Comparison of the Models

It is evident that classical ML currently maintains a significant advantage over QML in terms of runtime. Nonetheless, our experiments have revealed interesting patterns, which we want to summarize in the following. We will first begin by comparing the performances of the final classical and quantum models, answering our first research question, and finally, draw concrete recommendations for building QML models, thus answering the remaining research questions.

In addition to significantly longer fitting times, substantial differences in performance exist between classical ML models and their quantum counterparts. We were unable to achieve superior model performance using VQAs. Notably, both classical and quantum models exhibit a similar tendency in classification tasks, favoring the prediction of majority classes rather than considering all classes.

We believe this limitation was reinforced by the necessity of reducing the training sample size due to the excessive runtime of quantum models. It is widely recognized that a larger dataset often leads to improved ML model performance. Thus, using more samples could have potentially led to models that generalize better to unseen data.

When comparing different QML models across the datasets, certain consistent patterns can be observed despite the specific characteristics of each dataset. In the following, we will discuss these separately for each hyperparameter.

Optimizer One of the patterns which we found most consistently throughout all datasets was that Nelder-Mead had difficulties converging (as was depicted in Figure 5.39a). Not one Nelder-Mead configurations was among the best ones, and COBYLA and SPSA outperformed it on all experiments we conducted. We sometimes found significant differences for COBYLA and SPSA, however, overall neither one clearly outperformed

the other one. Even when we found significant differences across all configurations, the best configurations were usually made up of equal shares of both optimizers. However, given that COBYLA was usually a lot faster than SPSA, we would argue to employ COBYLA in the beginning and only switch to a different optimizer if problems arise.

Feature Map Another pattern we found consistently across all experiments was that ZFeatureMap was significantly better than ZZFeatureMap. Therefore, we believe that it is a good option to start with, and to only switch if the performance is bad. ZZFeatureMap could be a reasonable alternative, but we also want to point out that PauliFeatureMap is a highly customizable alternative. The Pauli gates can be specified freely, and tuning them properly could lead to significant performance gains. However, one can choose between four gates and concatenate them arbitrarily, so the search space grows quite quickly.

Feature Map Entanglement Should a different feature map than the ZFeatureMap be used, the entanglement strategy can be set as well. Our experiments did not give us clear insights into the strategies, however, we found 'full' entanglement to often be significantly outperformed by other strategies.

Ansatz In most of our experiments, we did not find significant differences between different ansatz choices. Nonetheless, PauliTwoDesign could consistently not be found among the best performing ansatzes, while also not being among the worst performing ones either. We found no significant differences among the other ansatzes, however, given that EfficientSU2 usually had a longer runtime, we would advise starting with either TwoLocal or RealAmplitudes.

TwoLocal is a highly customizable ansatz, and rotation and entanglement blocks have to be set accordingly. Again, tuning the one qubit rotation and two qubit entanglement gates accordingly might lead to significant performance gains. However, due to the vast possibilities of setting the blocks, the search space also grows quite quickly. We would therefore advise starting with the RealAmplitudes ansatz, as it can be used out-of-the-box, without having to think about rotation and entanglement blocks.

Ansatz Entanglement We found no significant differences regarding the ansatz entanglement consistently across our experiments, neither performance- nor runtime-wise. Choosing the default strategy and only tuning it should the performance not be sufficient seems like a viable option.

Preprocessing In the majority of our experiments, PCA significantly outperformed LDA. Still, for the cover type dataset, all the best configurations used LDA as the preprocessing step, and for the KDD dataset, the best configurations use PCA. We therefore cannot advise using one or the other, but rather recommend testing, which one works best beforehand.

We found that noise does not necessarily decrease the performance of the models overall, but often even increases it. However, it negatively affected the performance of the best-performing models throughout our tuning experiments. Nonetheless, our experiments did not reveal any clear patterns about which parameter choices are more and which ones are less affected by noise in the circuit.

CHAPTER 6

Conclusion

This thesis has explored quantum machine learning (QML), emphasizing the potential of quantum computing in the post-Moore age. The ability of quantum systems to process vast amounts of information and obtain significant algorithmic speed-ups has the potential to unlock new frontiers in machine learning (ML). When looking at the results and runtime, however, it is important to keep in mind that, while a lot of research has gone into classical ML for decades, the field of QML and quantum computing (QC) in general are still in early stages of development.

Several papers have been published that try to optimize the performance of a QML model on a single dataset. In this thesis, we have extended the approach, trying to find configurations that work well on a variety of problems, by comparing QML and classical ML models on four different datasets for classification and regression.

Our results suggest that classical ML still has a head start over QML, both in terms of runtime and performance. We found big performance differences across the datasets. One of the biggest limitations is, in our opinion, that the runtime of a variational quantum algorithm (VQA) quickly increases with more training data, hence, we had to tune the models using only 400 samples and train the final models on 5,000 data points. Both the classical and quantum models showed that they had problems predicting minority classes for classification and outliers for regression. The limited samples made it difficult to produce models that generalize.

Nonetheless, we have gained valuable insights into tuning parameters for VQAs. In particular, the choice of feature map and optimizer is crucial. COBYLA and SPSA seem to both be reasonable choices for the optimizer, while Nelder-Mead consistently underperforms. ZFeatureMap outperforms ZZFeatureMap across all our experiments. Among the entanglement strategies for ZZFeatureMap, we find that 'full' entanglement is often significantly worse than other strategies, and it might be favorable to use 'pairwise', 'linear' or 'sca'.

The ansatzes perform similarly on average, however, some are more likely to lead to higher-performing outliers. Therefore, we advise starting with RealAmplitudes, which can be used out-of-the-box and is often faster than EfficientSU2. Furthermore, TwoLocal is a good option as well, however, the entanglement and rotation gates have to be selected manually. We find hardly any significant differences between the chosen entanglement strategy, and therefore believe using the default option is a good start.

Regarding preprocessing for classification, we were not able to reproduce the results reported in [MP22, HQBCTC22]. In the majority of experiments, PCA was significantly better than LDA. Notably, however, the best configurations for the cover type dataset all used LDA, whereas PCA was favorable for the KDD Cup dataset.

Overall, there is an enormous amount of parameters one can tune in a VQA. For our exhaustive search, we had to restrict our search space to the most fundamental ones, which poses a limitation to our goal of finding configurations that lead to high performing models. Notably, we believe replacing the exhaustive search and switching to a more stochastic setup (such as an f-race) would allow testing more hyperparameters, while eliminating consistently underperforming ones already in the very beginning.

Furthermore, it would be interesting to investigate the impact of different levels of noise more thoroughly. Qiskit allows building complex noise models, and it would be fascinating to see how the different errors and error probabilities affect the performance of the models.

Finally, using only seven qubits to represent the data is a limitation as well. We believe it may be interesting to investigate how the performance changes with a higher number of qubits.

List of Figures

List of Tables

Acronyms

BP barren plateau. 26–28, 34, 66, 72

GB gradient boosting. 47, 56–58, 60, 61

K-NN k-nearest neighbors. 47, 58, 60

LDA linear discriminant analysis. 33, 50, 51, 81, 82, 91, 100, 104

MAE mean absolute error. 3, 54, 56, 58, 63, 66, 72, 74, 133

ML machine learning. 3, 4, 17–20, 22, 24, 27, 33, 37, 39, 47, 50, 55, 99, 103

MSE mean squared error. 3, 54, 56–58, 61, 63–69, 71–76, 133

NISQ Noisy Intermediate-Scale Quantum. 2, 10, 21–24, 29, 36

NN neural network. 18, 26, 27, 29, 35

PCA principal component analysis. 33, 50, 51, 81, 82, 91, 93, 100, 104

QC quantum computing. 1, 3, 5, 10, 11, 13, 15, 18, 32, 103

QM quantum mechanics. 5–8, 10

QML quantum machine learning. 1, 3–5, 17–21, 24, 27, 29, 31–33, 39, 48, 50, 51, 55, 56, 99, 103, 133

RF random forest. 47, 57, 58, 61

SVM support vector machine. 47, 58, 59

VQA variational quantum algorithm. 1, 3, 4, 17, 18, 21–24, 26–29, 31–36, 48, 50, 51, 99, 103, 104

Bibliography

[ABBB21] Leonardo Alchieri, Davide Badalotti, Pietro Bonardi, and Simone Bianco. An introduction to quantum machine learning: from quantum logic to quantum deep learning. *Quantum Machine Intelligence*, 3(2):28, December 2021.

[Ama98] Shun-ichi Amari. Natural Gradient Works Efficiently in Learning. *Neural Computation*, 10(2):251–276, February 1998.

[ASZ+21] Amira Abbas, David Sutter, Christa Zoufal, Aurelien Lucchi, Alessio Figalli, and Stefan Woerner. The power of quantum neural networks. *Nature Computational Science*, 1(6):403–409, June 2021.

[BBF+22] Aaron Baughman, Daniel Bohm, Micah Forster, Eduardo Morales, Jeff Powell, Shaun McPartlin, Raja Hebbar, Kavitha Yogaraj, Yoshika Chhabra, Sudeep Ghosh, Rukhsan Ul Haq, and Arjun Kashyap. Large Scale Diverse Combinatorial Optimization: ESPN Fantasy Football Player Trades, April 2022. arXiv:2111.02859 [cs].

[BCV+23] Matias Bilkis, Marco Cerezo, Guillaume Verdon, Patrick J. Coles, and Lukasz Cincio. A semi-agnostic ansatz with variable structure for quantum machine learning, January 2023. arXiv:2103.06712 [quant-ph, stat].

[Bia21] Jacob Biamonte. Universal variational quantum computation. *Physical Review A*, 103(3):L030401, March 2021.

[BK21] Lennart Bittel and Martin Kliesch. Training Variational Quantum Algorithms Is NP-Hard. *Physical Review Letters*, 127(12):120502, September 2021.

[BK22] Gregory Boyd and Bálint Koczor. Training Variational Quantum Circuits with CoVaR: Covariance Root Finding with Classical Shadows. *Physical Review X*, 12(4):041022, November 2022.

[Bli21] Seymour Michael Blinder. *Introduction to quantum mechanics*. Academic Press, London ; San Diego, CA, 2nd edition, revised and expanded edition, 2021. OCLC: on1197813826.

[BLSF19] Marcello Benedetti, Erika Lloyd, Stefan Sack, and Mattia Fiorentini. Parameterized quantum circuits as machine learning models. *Quantum Science and Technology*, 4(4):043001, November 2019.

[BMWV+23] Xavier Bonet-Monroig, Hao Wang, Diederick Vermetten, Bruno Senjean, Charles Moussa, Thomas Bäck, Vedran Dunjko, and Thomas E. O'Brien. Performance comparison of optimization methods on variational quantum algorithms. *Physical Review A*, 107(3):032407, March 2023.

[CAB+21] Marco Cerezo, Andrew Arrasmith, Ryan Babbush, Simon C. Benjamin, Suguru Endo, Keisuke Fujii, Jarrod R. McClean, Kosuke Mitarai, Xiao Yuan, Lukasz Cincio, and Patrick J. Coles. Variational quantum algorithms. *Nature Reviews Physics*, 3(9):625–644, September 2021.

[CCL23] Berta Casas and Alba Cervera-Lierta. Multi-dimensional Fourier series with quantum circuits, March 2023. arXiv:2302.03389 [quant-ph].

[CDPB+21] Alexandre Choquette, Agustin Di Paolo, Panagiotis Kl. Barkoutsos, David Sénéchal, Ivano Tavernelli, and Alexandre Blais. Quantum-optimal-control-inspired ansatz for variational quantum algorithms. *Physical Review Research*, 3(2):023092, May 2021.

[CEMM98] Richard Cleve, Artur Ekert, Chiara Macchiavello, and Michele Mosca. Quantum algorithms revisited. *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 454(1969):339–354, January 1998.

[CHI+18] Carlo Ciliberto, Mark Herbster, Alessandro Davide Ialongo, Massimiliano Pontil, Andrea Rocchetto, Simone Severini, and Leonard Wossnig. Quantum machine learning: a classical perspective. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 474(2209):20170551, January 2018.

[con23] Qiskit contributors. Qiskit: An Open-source Framework for Quantum Computing, 2023.

[CRAG18] Yudong Cao, Jhonathan Romero, and Alán Aspuru-Guzik. Potential of quantum computing for drug discovery. *IBM Journal of Research and Development*, 62(6):2:6:1–2:6:20, November 2018.

[CRO+19] Yudong Cao, Jhonathan Romero, Jonathan P. Olson, Matthias Degroote, Peter D. Johnson, Mária Kieferová, Ian D. Kivlichan, Tim Menke, Borja Peropadre, Nicolas P. D. Sawaya, Sukin Sim, Libor Veis, and Alán Aspuru-Guzik. Quantum Chemistry in the Age of Quantum Computing. *Chemical Reviews*, 119(19):10856–10915, October 2019.

[CSV+21] Marco Cerezo, Akira Sone, Tyler Volkoff, Lukasz Cincio, and Patrick J. Coles. Cost function dependent barren plateaus in shallow parametrized quantum circuits. *Nature Communications*, 12(1):1791, March 2021.

[CYQ+20] Samuel Yen-Chi Chen, Chao-Han Huck Yang, Jun Qi, Pin-Yu Chen, Xiaoli Ma, and Hsi-Sheng Goan. Variational Quantum Circuits for Deep Reinforcement Learning. *IEEE Access*, 8:141007–141024, 2020.

[Das22] Annika Daspal. OptiPauli: An algorithm to find a near-optimal Pauli Feature Map for Quantum Support Vector Classifiers. In *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 828–830, September 2022.

[Deu85] David Deutsch. Quantum theory, the Church–Turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 400(1818):97–117, July 1985.

[DG17] Dheeru Dua and Casey Graff. UCI Machine Learning Repository, 2017.

[DHY+22] Yuxuan Du, Tao Huang, Shan You, Min-Hsiu Hsieh, and Dacheng Tao. Quantum circuit architecture search for variational quantum algorithms. *npj Quantum Information*, 8(1):62, May 2022.

[Dir58] Paul A.M. Dirac. *The Principles of Quantum Mechanics*. Oxford University Press, 4 edition, 1958.

[DiV95] David P. DiVincenzo. Two-bit gates are universal for quantum computation. *Physical Review A*, 51(2):1015–1022, February 1995.

[DJ92] David Deutsch and Richard Jozsa. Rapid solution of problems by quantum computation. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, 439(1907):553–558, December 1992.

[DL21] Gennaro De Luca. Survey of NISQ Era Hybrid Quantum-Classical Machine Learning Research. *Journal of Artificial Intelligence and Technology*, December 2021.

[DMH+18] Eugene F. Dumitrescu, Alex J. McCaskey, Gaute Hagen, Gustav R. Jansen, Titus D. Morris, Thomas Papenbrock, Raphael C. Pooser, David Jarvis Dean, and Pavel Lougovski. Cloud Quantum Computing of an Atomic Nucleus. *Physical Review Letters*, 120(21):210501, May 2018.

[DTYT22] Yuxuan Du, Zhuozhuo Tu, Xiao Yuan, and Dacheng Tao. Efficient Measure for the Expressivity of Variational Quantum Algorithms. *Physical Review Letters*, 128(8):080506, February 2022.

[EPR35] Albert Einstein, Boris Podolsky, and Nathan Rosen. Can Quantum-Mechanical Description of Physical Reality Be Considered Complete? *Physical Review*, 47(10):777–780, May 1935.

[FGG14] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A Quantum Approximate Optimization Algorithm, November 2014. arXiv:1411.4028 [quant-ph].

[GCA22] Amir H. Gandomi, Fang Chen, and Laith Abualigah. Machine Learning Technologies for Big Data Analytics. *Electronics*, 11(3):421, January 2022.

[GEBM19] Harper R. Grimsley, Sophia E. Economou, Edwin Barnes, and Nicholas J. Mayhall. An adaptive variational algorithm for exact molecular simulations on a quantum computer. *Nature Communications*, 10(1):3007, July 2019.

[GI19] Laszlo Gyongyosi and Sandor Imre. A Survey on quantum computing technology. *Computer Science Review*, 31:51–71, February 2019.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing - STOC '96*, pages 212–219, Philadelphia, Pennsylvania, United States, 1996. ACM Press.

[GTN21] Takahiro Goto, Quoc Hoan Tran, and Kohei Nakajima. Universal Approximation Property of Quantum Machine Learning Models in Quantum-Enhanced Feature Spaces. *Physical Review Letters*, 127(9):090506, August 2021.

[HCT+19] Vojtěch Havlíček, Antonio D. Córcoles, Kristan Temme, Aram W. Harrow, Abhinav Kandala, Jerry M. Chow, and Jay M. Gambetta. Supervised learning with quantum-enhanced feature spaces. *Nature*, 567(7747):209–212, March 2019.

[HD21] Patrick Huembeli and Alexandre Dauphin. Characterizing the loss landscape of variational quantum circuits. *Quantum Science and Technology*, 6(2):025011, April 2021.

[HHL09] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum Algorithm for Linear Systems of Equations. *Physical Review Letters*, 103(15):150502, October 2009.

[HQBCTC22] Juan Kenyhy Hancco-Quispe, Jordan Piero Borda-Colque, and Fred Torres-Cruz. Quantum Machine Learning Applied to the Classification of Diabetes, December 2022. arXiv:2301.00109 [cs].

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, January 1989.

[IYLV20] Artur F. Izmaylov, Tzu-Ching Yen, Robert A. Lang, and Vladyslav Verteletskyi. Unitary Partitioning Approach to the Measurement Problem in the Variational Quantum Eigensolver Method. *Journal of Chemical Theory and Computation*, 16(1):190–195, January 2020.

[JKA21a] Nisheeth Joshi, Pragya Katyayan, and Syed Afroz Ahmed. Comparing Classical ML Models with Quantum ML Models with Parametrized Circuits for Sentiment Analysis Task. *Journal of Physics: Conference Series*, 1854(1):012032, April 2021.

[JKA21b] Nisheeth Joshi, Pragya Katyayan, and Syed Afroz Ahmed. Evaluating the Performance of Some Local Optimizers for Variational Quantum Classifiers. *Journal of Physics: Conference Series*, 1817(1):012015, March 2021.

[JRO+17] Peter D. Johnson, Jonathan Romero, Jonathan Olson, Yudong Cao, and Alán Aspuru-Guzik. QVECTOR: an algorithm for device-tailored quantum error correction, November 2017. arXiv:1711.02249 [quant-ph].

[KB15] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *International Conference for Learning Representations*, 3, 2015.

[Kit95] Alexei Kitaev. Quantum measurements and the Abelian Stabilizer Problem. *arXiv:quant-ph/9511026*, 1995.

[KJ23] Pragya Katyayan and Nisheeth Joshi. Supervised Question Classification on SelQA Dataset Using Variational Quantum Classifiers. In Deepak Gupta, Ashish Khanna, Aboul Ella Hassanien, Sameer Anand, and Ajay Jaiswal, editors, *International Conference on Innovative Computing and Communications*, Lecture Notes in Networks and Systems, pages 695–706, Singapore, 2023. Springer Nature.

[KPB97] R. Kelley Pace and Ronald Barry. Sparse spatial autoregressions. *Statistics & Probability Letters*, 33(3):291–297, May 1997.

[LBR17] Austin Lund, Michael J. Bremner, and Timothy C. Ralph. Quantum sampling problems, BosonSampling and quantum supremacy. *npj Quantum Information*, 3(1):1–8, April 2017.

[LC20] Ryan LaRose and Brian Coyle. Robust data encodings for quantum classifiers. *Physical Review A*, 102(3):032420, September 2020.

[LHF+00] Richard Lippmann, Joshua W Haines, David J Fried, Jonathan Korba, and Kumar Das. The 1999 DARPA off-line intrusion detection evaluation. *Computer Networks*, 34(4):579–595, October 2000.

[LKL+19] Hsuan-Hao Lu, Natalie Klco, Joseph M. Lukens, Titus D. Morris, Aaina Bansal, Andreas Ekström, Gaute Hagen, Thomas Papenbrock, Andrew M. Weiner, Martin J. Savage, and Pavel Lougovski. Simulations of subatomic many-body physics on a quantum frequency processor. *Physical Review A*, 100(1):012320, July 2019.

[LSI+20] Seth Lloyd, Maria Schuld, Aroosa Ijaz, Josh Izaac, and Nathan Killoran. Quantum embeddings for machine learning, February 2020. arXiv:2001.03622 [quant-ph].

[Mar14] Igor L. Markov. Limits on fundamental limits to computation. *Nature*, 512(7513):147–154, August 2014.

[MBS+18] Jarrod R. McClean, Sergio Boixo, Vadim N. Smelyanskiy, Ryan Babbush, and Hartmut Neven. Barren plateaus in quantum neural network training landscapes. *Nature Communications*, 9(1):4812, November 2018.

[Mer07] N. David Mermin. *Quantum Computer Science: An Introduction*. Cambridge University Press, 1 edition, August 2007.

[MKR+21] Nimish Mishra, Manik Kapil, Hemant Rakesh, Amit Anand, Nilima Mishra, Aakash Warke, Soumya Sarkar, Sanchayan Dutta, Sabhyata Gupta, Aditya Prasad Dash, Rakshit Gharat, Yagnik Chatterjee, Shuvarati Roy, Shivam Raj, Valay Kumar Jain, Shreeram Bagaria, Smit Chaudhary, Vishwanath Singh, Rituparna Maji, Priyanka Dalei, Bikash K. Behera, Sabyasachi Mukhopadhyay, and Prasanta K. Panigrahi. Quantum Machine Learning: A Review and Current Status. In *Data Management, Analytics and Innovation*, Advances in Intelligent Systems and Computing, pages 101–145, Singapore, 2021. Springer.

[MP22] Javier Mancilla and Christophe Pere. A Preprocessing Perspective for Quantum Machine Learning Classification Advantage in Finance Using NISQ Algorithms. *Entropy*, 24(11):1656, November 2022.

[NM65] John A. Nelder and Roger Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 7(4):308–313, January 1965.

[OM20] Wesley O'Quinn and Shiwen Mao. Quantum Machine Learning: Recent Advances and Outlook. *IEEE Wireless Communications*, 27(3):126–131, June 2020.

[OMACAC+23] Emmanuel Ovalle-Magallanes, Dora E. Alvarado-Carrillo, Juan Gabriel Avina-Cervantes, Ivan Cruz-Aceves, and Jose Ruiz-Pinales. Quantum

angle encoding with learnable rotation applied to quantum–classical convolutional neural networks. *Applied Soft Computing*, 141:110307, July 2023.

[OSS+21] Carlos Outeiral, Martin Strahm, Jiye Shi, Garrett M. Morris, Simon C. Benjamin, and Charlotte M. Deane. The prospects of quantum computing in computational molecular biology. *WIREs Computational Molecular Science*, 11(1), January 2021.

[OTM+21] Mateusz Ostaszewski, Lea M. Trenkwalder, Wojciech Masarczyk, Eleanor Scerri, and Vedran Dunjko. Reinforcement learning for optimization of variational quantum circuit architectures. In *Advances in Neural Information Processing Systems*, volume 34, pages 18182–18194. Curran Associates, Inc., 2021.

[Pas23] Davide Pastorello. *Concise guide to quantum machine learning*. Springer, Singapore, 2023. OCLC: 1362515386.

[Pow94] Michael J. D. Powell. A Direct Search Optimization Method That Models the Objective and Constraint Functions by Linear Interpolation. In Susana Gomez and Jean-Pierre Hennart, editors, *Advances in Optimization and Numerical Analysis*, Mathematics and Its Applications, pages 51–67. Springer Netherlands, Dordrecht, 1994.

[PR22] Ilya Piatrenka and Marian Rusek. Quantum Variational Multi-class Classifier for the Iris Data Set. In Derek Groen, Clélia de Mulatier, Maciej Paszynski, Valeria V. Krzhizhanovskaya, Jack J. Dongarra, and Peter M. A. Sloot, editors, *Computational Science – ICCS 2022*, Lecture Notes in Computer Science, pages 247–260, Cham, 2022. Springer International Publishing.

[Pre18] John Preskill. Quantum Computing in the NISQ era and beyond. *Quantum*, 2:79, August 2018.

[PVG+11] Fabian Pedregose, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Mathieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine Learning in {P}ython. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[PWK22] Hrushikesh Patil, Yulun Wang, and Predrag S. Krstić. Variational quantum linear solver with a dynamic ansatz. *Physical Review A*, 105(1):012423, January 2022.

[RDHAB21] Javier Rivera-Dean, Patrick Huembeli, Antonio Acín, and Joseph Bowles. Avoiding local minima in Variational Quantum Algorithms with Neural Networks, October 2021. arXiv:2104.02955 [quant-ph].

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, October 1986.

[RK21] Salonik Resch and Ulya R. Karpuzcu. Benchmarking Quantum Computers and the Impact of Quantum Noise. *ACM Computing Surveys*, 54(7):142:1–142:35, July 2021.

[RLC+20] Alessandro Roggero, Andy C. Y. Li, Joseph Carlson, Rajan Gupta, and Gabriel N. Perdue. Quantum computing for neutrino-nucleus scattering. *Physical Review D*, 101(7):074038, April 2020.

[RML14] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. Quantum Support Vector Machine for Big Data Classification. *Physical Review Letters*, 113(13):130503, September 2014.

[RRC19] Gopinath Rebala, Ajay Ravi, and Sanjay Churiwala. *An introduction to machine learning*. Springer Science+Business Media, New York, NY, 1 edition, 2019.

[RSA78] Ronald L. Rivest, Adi Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[RSL22] Ali Rad, Alireza Seif, and Norbert M. Linke. Surviving The Barren Plateau in Variational Quantum Circuits with Bayesian Learning Initialization. *arXiv:2203.02464 [quant-ph]*, 2022.

[SBG+19] Maria Schuld, Ville Bergholm, Christian Gogolin, Josh Izaac, and Nathan Killoran. Evaluating analytic gradients on quantum hardware. *Physical Review A*, 99(3):032331, March 2019.

[Sch03] Ralf Schützhold. Pattern recognition on a quantum computer. *Physical Review A*, 67(6):062311, June 2003.

[SGF19] Giuseppe Sergioli, Roberto Giuntini, and Hector Freytes. A new quantum approach to binary classification. *PLOS ONE*, 14(5):e0216224, May 2019.

[Sha20] John Shalf. The future of computing beyond Moore's Law. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 378(2166):20190061, March 2020.

[Sho97] Peter W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Journal on Computing*, 26(5):1484–1509, October 1997.

[SJY20] Veerappampalayam E. Sathishkumar, Park Jangwoo, and Cho Yongyun. Using data mining techniques for bike sharing demand prediction in metropolitan city. *Computer Communications*, 153:353–366, March 2020.

[SK19] Maria Schuld and Nathan Killoran. Quantum Machine Learning in Feature Hilbert Spaces. *Physical Review Letters*, 122(4):040504, February 2019.

[SK22] Maria Schuld and Nathan Killoran. Is Quantum Advantage the Right Goal for Quantum Machine Learning? *PRX Quantum*, 3(3):030101, July 2022.

[SM22] Hatma Suryotrisongko and Yasuo Musashi. Hybrid Quantum Deep Learning and Variational Quantum Classifier-Based Model for Botnet DGA Attack Detection. *International Journal of Intelligent Engineering and Systems*, 15(3):215–224, June 2022.

[Spa98] James Spall. An Overview of the Simultaneous Perturbation Method for Efficient Optimization. *Johns Hopkins Apl Technical Digest*, 19(4):482–492, 1998.

[SSM21] Maria Schuld, Ryan Sweke, and Johannes Jakob Meyer. Effect of data encoding on the expressive power of variational quantum-machine-learning models. *Physical Review A*, 103(3):032430, March 2021.

[SSPT23] Daniel Sierra-Sosa, Soham Pal, and Michael Telahun. Data rotation and its influence on quantum encoding. *Quantum Information Processing*, 22(1):89, January 2023.

[SY20] Veerappampalayam E. Sathishkumar and Cho Yongyun. A rule-based model for Seoul Bike sharing demand prediction using weather data. *European Journal of Remote Sensing*, 53(sup1):166–183, June 2020.

[SYG+20] Yudai Suzuki, Hiroshi Yano, Qi Gao, Shumpei Uno, Tomoki Tanaka, Manato Akiyama, and Naoki Yamamoto. Analysis and synthesis of feature map for kernel-based quantum classifier. *Quantum Machine Intelligence*, 2(1):9, July 2020.

[SYH+20] Kevin J Sung, Jiahao Yao, Matthew P Harrigan, Nicholas C Rubin, Zhang Jiang, Lin Lin, Ryan Babbush, and Jarrod R McClean. Using models to improve optimizers for variational quantum algorithms. *Quantum Science and Technology*, 5(4):044008, October 2020.

[TGIH17] Alaa Tharwat, Tarek Gaber, Abdelhameed Ibrahim, and Aboul Ella Hassanien. Linear discriminant analysis: A detailed tutorial. *AI Communications*, 30(2):169–190, May 2017.

[TLI+23] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and Efficient Foundation Language Models. *arXiv:2302.13971 [cs.CL]*, 2023.

[TMGB19] Francesco Tacchino, Chiara Macchiavello, Dario Gerace, and Daniele Bajoni. An artificial neuron implemented on an actual quantum processor. *npj Quantum Information*, 5(1):26, March 2019.

[Vil86] C N Villars. The paradox of Schrodinger's cat. *Physics Education*, 21(4):232–237, July 1986.

[WFC+21] Samson Wang, Enrico Fontana, M. Cerezo, Kunal Sharma, Akira Sone, Lukasz Cincio, and Patrick J. Coles. Noise-induced barren plateaus in variational quantum algorithms. *Nature Communications*, 12(1):6961, November 2021.

[WZ82] William K. Wootters and Wojciech H. Zurek. A single quantum cannot be cloned. *Nature*, 299(5886):802–803, October 1982.

[XNJR02] Eric P. Xing, Andrew Y. Ng, Michael I. Jordan, and Stuart Russell. Distance metric learning, with application to clustering with side-information. In *Proceedings of the 15th International Conference on Neural Information Processing Systems*, NIPS'02, pages 521–528, Cambridge, MA, USA, January 2002. MIT Press.

[ZTK+20] Andrew Zhao, Andrew Tranter, William M. Kirby, Shu Fay Ung, Akimasa Miyake, and Peter J. Love. Measurement reduction in variational quantum algorithms. *Physical Review A*, 101(6):062322, June 2020.

APPENDIX A

Additional Mathematical Explanations

Equation A.1 describes a calculation step that was skipped in Section 2.2.5. We add it here for clarification, but note that it is just a mere multiplication and tensor product of the different gates.

$$
\begin{aligned}
&(H \otimes 1) * (\frac{1}{2} * (|0\rangle + |1\rangle) * (|f(0)\rangle - |f\tilde{(}0)\rangle) = \\
&(H \otimes 1) * (\frac{1}{2} * (|0\rangle|f(0)\rangle - |0\rangle|f\tilde{(}0)\rangle + |1\rangle|f(0)\rangle - |1\rangle|f\tilde{(}0)\rangle) = \\
&\frac{1}{2} * (H|0\rangle \otimes 1|f(0)\rangle - H|0\rangle \otimes 1|f\tilde{(}0)\rangle + H|1\rangle \otimes 1|f(0)\rangle - H|1\rangle \otimes 1|f\tilde{(}0)\rangle) = \\
&\frac{1}{2}(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |f(0)\rangle - \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |f\tilde{(}0)\rangle + \\
&\quad \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \otimes |f(0)\rangle - \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \otimes |f\tilde{(}0)\rangle = \\
&\frac{1}{2}(\frac{1}{\sqrt{2}}(2|0\rangle \otimes |f(0)\rangle - 2|0\rangle \otimes |f\tilde{(}0)\rangle)) = \\
&\frac{1}{\sqrt{2}}|0\rangle(|f(0)\rangle - |f\tilde{(}0)\rangle)
\end{aligned}
\tag{A.1}
$$

APPENDIX B

Data Summary Statistics

This chapter contains summary statistics of the datasets as described in Section 4.1.

	MedInc	HouseAge	AvgRooms	AvgBedrms	Population	AvgOccup	Lat.	Lon.	Price
mean	3.87	28.64	5.43	1.10	1425.48	3.07	35.63	-119.57	2.07
std	1.90	12.59	2.47	0.47	1132.46	10.39	2.14	2.00	1.15
min	0.50	1.00	0.85	0.33	3.00	0.69	32.54	-124.35	0.15
25%	2.56	18.00	4.44	1.01	787.00	2.43	33.93	-121.80	1.20
50%	3.53	29.00	5.23	1.05	1166.00	2.82	34.26	-118.49	1.80
75%	4.74	37.00	6.05	1.10	1725.00	3.28	37.71	-118.01	2.65
max	15.00	52.00	141.91	34.07	35682.00	1243.33	41.95	-114.31	5.00

Table B.1: Summary Statistics: California Housing

	Bikes	Hour	Temp.	Hum.	Wind	Visibility	Dew point	Radiation	Rain	Snow
mean	704.60	11.50	12.88	58.23	1.72	1436.83	4.07	0.57	0.15	0.08
std	645.00	6.92	11.94	20.36	1.04	608.30	13.06	0.87	1.13	0.44
min	0.00	0.00	-17.80	0.00	0.00	27.00	-30.60	0.00	0.00	0.00
25%	191.00	5.75	3.50	42.00	0.90	940.00	-4.70	0.00	0.00	0.00
50%	504.50	11.50	13.70	57.00	1.50	1698.00	5.10	0.01	0.00	0.00
75%	1065.25	17.25	22.50	74.00	2.30	2000.00	14.80	0.93	0.00	0.00
max	3556.00	23.00	39.40	98.00	7.40	2000.00	27.20	3.52	35.00	8.80

Table B.2: Summary Statistics: Seoul Bike Sharing

	Elev.	Aspect	Slope	H.Hyd.	V.Hyd.	H.Road.	Hill.9	Hill.12	Hill.3	H.Fire.	Cov.	Wild.	Soil
mean	2959.37	155.66	14.10	269.43	46.42	2350.15	212.15	223.32	142.53	1980.29	2.05	2.11	23.36
std	279.98	111.91	7.49	212.55	58.30	1559.25	26.77	19.77	38.27	1324.20	1.40	1.06	9.49
min	1859.00	0.00	0.00	0.00	-173.00	0.00	0.00	0.00	0.00	0.00	1.00	1.00	0.00
25%	2809.00	58.00	9.00	108.00	7.00	1106.00	198.00	213.00	119.00	1024.00	1.00	1.00	19.00
50%	2996.00	127.00	13.00	218.00	30.00	1997.00	218.00	226.00	143.00	1710.00	2.00	2.00	28.00
75%	3163.00	260.00	18.00	384.00	69.00	3328.00	231.00	237.00	168.00	2550.00	2.00	3.00	30.00
max	3858.00	360.00	66.00	1397.00	601.00	7117.00	254.00	254.00	254.00	7173.00	7.00	4.00	39.00

Table B.3: Summary Statistics: Cover Type

	duration	src_bytes	dst_bytes	land	wr_frag.	urgent	hot	failed_logins	logged_in	num_compr.
mean	47.98	3.025610e+03	868.53	0.00	0.01	0.00	0.03	0.00	0.15	0.01
std	707.75	9.882181e+05	33040.00	0.01	0.13	0.01	0.78	0.02	0.36	1.80
min	0.00	0.000000e+00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
25%	0.00	4.500000e+01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
50%	0.00	5.200000e+02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
75%	0.00	1.032000e+03	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
max	58329.00	6.933756e+08	5155468.00	1.00	3.00	3.00	30.00	5.00	1.00	884.00

Table B.4: Summary Statistics: KDD Cup 1999/1

	root_shell	su	root	file_creat.	num_shells	access_files	guest_login	count	srv_count	serror_rate
mean	0.00	0.00	0.01	0.0	0.00	0.00	0.00	332.29	292.91	0.18
std	0.01	0.01	2.01	0.1	0.01	0.04	0.04	213.15	246.32	0.38
min	0.00	0.00	0.00	0.0	0.00	0.00	0.00	0.00	0.00	0.00
25%	0.00	0.00	0.00	0.0	0.00	0.00	0.00	117.00	10.00	0.00
50%	0.00	0.00	0.00	0.0	0.00	0.00	0.00	510.00	510.00	0.00
75%	0.00	0.00	0.00	0.0	0.00	0.00	0.00	511.00	511.00	0.00
max	1.00	2.00	993.00	28.0	2.00	8.00	1.00	511.00	511.00	1.00

Table B.5: Summary Statistics: KDD Cup 1999/2

	srv_serror	rerror	srv_rerror	same_srv	diff_srv	srv_diff_host	dst	dst_srv	dst_same	dst_diff
mean	0.18	0.06	0.06	0.79	0.02	0.03	232.47	188.67	0.75	0.03
std	0.38	0.23	0.23	0.39	0.08	0.14	64.75	106.04	0.41	0.11
min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
25%	0.00	0.00	0.00	1.00	0.00	0.00	255.00	46.00	0.41	0.00
50%	0.00	0.00	0.00	1.00	0.00	0.00	255.00	255.00	1.00	0.00
75%	0.00	0.00	0.00	1.00	0.00	0.00	255.00	255.00	1.00	0.04
max	1.00	1.00	1.00	1.00	1.00	1.00	255.00	255.00	1.00	1.00

Table B.6: Summary Statistics: KDD Cup 1999/3

	dst_same_src_port	dst_srv_diff_host	dst_serror	dst_srv_serror	dst_rerror	dst_srv_rerror
mean	0.60	0.01	0.18	0.18	0.06	0.06
std	0.48	0.04	0.38	0.38	0.23	0.23
min	0.00	0.00	0.00	0.00	0.00	0.00
25%	0.00	0.00	0.00	0.00	0.00	0.00
50%	1.00	0.00	0.00	0.00	0.00	0.00
75%	1.00	0.00	0.00	0.00	0.00	0.00
max	1.00	1.00	1.00	1.00	1.00	1.00

Table B.7: Summary Statistics: KDD Cup 1999/4

APPENDIX C

Classical Machine Learning

This chapter contains the results of the classical machine learning models. These are described and interpreted in Section 5.2. The time is measured in seconds.

Model	Scaler	MSE	MAE	Time	Value
Gradient Boosting	MinMax	0.02	0.09	19.62	criterion: friedman_mse
				19.62	max_depth: 4
				19.62	max_features: sqrt
				19.62	n_estimators: 300
	-	0.02	0.09	14.15	criterion: squared_error
				14.15	max_depth: 4
				14.15	max_features: sqrt
				14.15	n_estimators: 300
	Standard	0.02	0.09	19.79	criterion: squared_error
				19.79	max_depth: 4
				19.79	max_features: sqrt
				19.79	n_estimators: 300
Random Forest	-	0.02	0.10	18.81	criterion: friedman_mse
				18.81	max_depth: 10
				18.81	max_features: log2
				18.81	n_estimators: 250
Linear Regression	Standard	0.02	0.10	0.01	fit_intercept: True
	MinMax	0.02	0.10	0.01	fit_intercept: True
	-	0.02	0.10	2.07	fit_intercept: True
Random Forest	Standard	0.02	0.10	24.18	criterion: friedman_mse
				24.18	max_depth: 8
				24.18	max_features: log2
				24.18	n_estimators: 250
	MinMax	0.02	0.10	23.94	criterion: friedman_mse
				23.94	max_depth: 8
				23.94	max_features: log2
				23.94	n_estimators: 200
Support Vector Machine	Standard	0.02	0.10	0.59	C: 1
				0.59	kernel: linear
	-	0.02	0.10	667.86	C: 0.01
				667.86	kernel: linear
	MinMax	0.02	0.11	0.42	C: 1
				0.42	kernel: linear
K Nearest Neighbors	Standard	0.02	0.11	0.16	algorithm: brute
				0.16	n_neighbors: 10
				0.16	weights: distance
	MinMax	0.02	0.11	0.20	algorithm: brute
				0.20	n_neighbors: 10
				0.20	weights: distance
	-	0.06	0.18	0.46	algorithm: auto
				0.46	n_neighbors: 30
				0.46	weights: uniform

Table C.1: Classical Baseline: California Housing

Model	Scaler	MSE	MAE	Time	Value
Gradient Boosting	-	0.01	0.06	12.99	criterion: squared_error
				12.99	max_depth: 2
				12.99	max_features: sqrt
				12.99	n_estimators: 300
	MinMax	0.01	0.06	14.56	criterion: squared_error
				14.56	max_depth: 2
				14.56	max_features: sqrt
				14.56	n_estimators: 300
	Standard	0.01	0.06	14.90	criterion: squared_error
				14.90	max_depth: 2
				14.90	max_features: sqrt
				14.90	n_estimators: 300
Random Forest	MinMax	0.01	0.06	20.43	criterion: friedman_mse
				20.43	max_depth: 10
				20.43	max_features: sqrt
				20.43	n_estimators: 250
	Standard	0.01	0.06	20.41	criterion: friedman_mse
				20.41	max_depth: 10
				20.41	max_features: sqrt
				20.41	n_estimators: 250
	-	0.01	0.06	16.87	criterion: friedman_mse
				16.87	max_depth: 10
				16.87	max_features: sqrt
				16.87	n_estimators: 250
K Nearest Neighbors	MinMax	0.01	0.07	0.17	algorithm: auto
				0.17	n_neighbors: 10
				0.17	weights: distance
	Standard	0.01	0.07	0.18	algorithm: auto
				0.18	n_neighbors: 10
				0.18	weights: distance
Support Vector Machine	Standard	0.01	0.08	0.57	C: 10
				0.57	gamma: 0.01
				0.57	kernel: rbf
	MinMax	0.01	0.08	0.27	C: 10
				0.27	gamma: 0.1
				0.27	kernel: rbf
Linear Regression	Standard	0.02	0.10	0.01	fit_intercept: True
	MinMax	0.02	0.10	0.01	fit_intercept: True
	-	0.02	0.10	1.90	fit_intercept: True
Support Vector Machine	-	0.02	0.10	313.61	C: 0.1
				313.61	kernel: linear
K Nearest Neighbors	-	0.02	0.11	0.47	algorithm: auto
				0.47	n_neighbors: 30
				0.47	weights: distance

Table C.2: Classical Baseline: Seoul Bike Sharing

Model	Scaler	Acc.	F-1	Time	Value
Support Vector Machine	Standard	0.69	0.69	0.34	C: 1
				0.34	kernel: linear
Logistic Regression	MinMax	0.65	0.65	0.33	C: 0.001
				0.33	penalty: None
Gradient Boosting	Standard	0.65	0.64	146.78	criterion: friedman_mse
				146.78	max_depth: 2
				146.78	max_features: sqrt
				146.78	n_estimators: 250
	-	0.65	0.64	146.47	criterion: friedman_mse
				146.47	max_depth: 2
				146.47	max_features: sqrt
				146.47	n_estimators: 250
Logistic Regression	Standard	0.64	0.64	0.37	C: 10
				0.37	penalty: l2
Gradient Boosting	MinMax	0.64	0.63	150.62	criterion: friedman_mse
				150.62	max_depth: 10
				150.62	max_features: sqrt
				150.62	n_estimators: 200
Support Vector Machine	-	0.64	0.63	342.96	C: 0.01
				342.96	kernel: linear
	MinMax	0.62	0.62	0.26	C: 10
				0.26	kernel: linear
Random Forest	-	0.63	0.61	14.97	criterion: gini
				14.97	max_depth: 8
				14.97	max_features: log2
				14.97	n_estimators: 100
	Standard	0.63	0.61	14.87	criterion: gini
				14.87	max_depth: 8
				14.87	max_features: log2
				14.87	n_estimators: 100
	MinMax	0.63	0.61	14.84	criterion: gini
				14.84	max_depth: 8
				14.84	max_features: log2
				14.84	n_estimators: 100
Logistic Regression	-	0.58	0.57	1.98	C: 0.01
				1.98	penalty: l2
K Nearest Neighbors	Standard	0.57	0.55	0.39	algorithm: auto
				0.39	n_neighbors: 15
				0.39	weights: distance
	MinMax	0.56	0.54	0.27	algorithm: auto
				0.27	n_neighbors: 15
				0.27	weights: uniform
	-	0.56	0.53	0.47	algorithm: auto
				0.47	n_neighbors: 25
				0.47	weights: distance

Table C.3: Classical Baseline: Cover Type

Model	Scaler	Acc.	F-1	Time	Value
Random Forest	-	0.98	0.98	13.58	criterion: entropy
				13.58	max_depth: 8
				13.58	max_features: log2
				13.58	n_estimators: 100
	Standard	0.98	0.98	13.40	criterion: entropy
				13.40	max_depth: 8
				13.40	max_features: log2
				13.40	n_estimators: 100
Gradient Boosting	-	0.98	0.97	66.86	criterion: friedman_mse
				66.86	max_depth: 2
				66.86	max_features: sqrt
				66.86	n_estimators: 200
	Standard	0.98	0.97	66.77	criterion: friedman_mse
				66.77	max_depth: 2
				66.77	max_features: sqrt
				66.77	n_estimators: 200
	MinMax	0.98	0.97	66.75	criterion: friedman_mse
				66.75	max_depth: 2
				66.75	max_features: sqrt
				66.75	n_estimators: 200
Random Forest	MinMax	0.98	0.97	13.54	criterion: gini
				13.54	max_depth: 8
				13.54	max_features: log2
				13.54	n_estimators: 250
Support Vector Machine	-	0.97	0.96	1.65	C: 0.1
				1.65	kernel: linear
Logistic Regression	Standard	0.97	0.96	0.33	C: 0.1
				0.33	penalty: l2
K Nearest Neighbors	-	0.97	0.96	0.55	algorithm: auto
				0.55	n_neighbors: 5
				0.55	weights: distance
Logistic Regression	-	0.97	0.96	2.06	C: 0.01
				2.06	penalty: l2
	MinMax	0.96	0.96	0.25	C: 0.001
				0.25	penalty: None
Support Vector Machine	Standard	0.96	0.96	0.18	C: 1
				0.18	kernel: linear
K Nearest Neighbors	Standard	0.96	0.95	0.30	algorithm: auto
				0.30	n_neighbors: 10
				0.30	weights: distance
Support Vector Machine	MinMax	0.97	0.95	0.16	C: 0.1
				0.16	kernel: linear
K Nearest Neighbors	MinMax	0.96	0.95	0.29	algorithm: auto
				0.29	n_neighbors: 10
				0.29	weights: distance

Table C.4: Classical Baseline: KDD Cup 1999

APPENDIX D

QML Hyperparameter Tuning

The following contains the top 40 configurations from the QML experiments. The following abbreviations are used. The time is measured in seconds.

- MSE: mean squared error
- MAE: mean absolute error
- Opt.: optimizer
- Feat.: feature map
- Ent.: ansatz entanglement
- Feat.Ent.: feature map entanglement
- Prep.: preprocessing

MSE	MAE	Time	Ansatz	Opt.	Feat.	Ent.	Feat.Ent.
0.046	0.176	31843.802	EfficientSU2	SPSA	Z	full	-
0.051	0.176	26506.437	EfficientSU2	SPSA	Z	sca	-
0.054	0.177	15496.522	EfficientSU2	COBYLA	Z	circular	-
0.055	0.185	21318.924	RealAmplitudes	SPSA	Z	sca	-
0.056	0.194	26568.813	EfficientSU2	SPSA	Z	circular	-
0.056	0.189	21693.524	TwoLocal	SPSA	Z	sca	-
0.056	0.191	21300.711	RealAmplitudes	SPSA	Z	circular	-
0.056	0.186	6846.013	TwoLocal	COBYLA	Z	full	-
0.057	0.187	5236.379	RealAmplitudes	COBYLA	Z	sca	-
0.057	0.192	21384.019	TwoLocal	SPSA	Z	circular	-
0.059	0.194	26908.049	RealAmplitudes	SPSA	Z	full	-
0.060	0.188	5692.743	RealAmplitudes	COBYLA	Z	circular	-
0.065	0.198	27078.418	TwoLocal	SPSA	Z	full	-
0.071	0.203	19527.893	EfficientSU2	COBYLA	Z	full	-
0.075	0.211	21017.120	TwoLocal	SPSA	Z	pairwise	-
0.077	0.213	21262.278	TwoLocal	SPSA	Z	linear	-
0.078	0.216	25924.091	EfficientSU2	SPSA	Z	linear	-
0.080	0.217	5868.035	TwoLocal	COBYLA	Z	linear	-
0.084	0.210	5788.658	TwoLocal	COBYLA	Z	sca	-
0.085	0.217	21081.794	RealAmplitudes	SPSA	Z	linear	-
0.086	0.217	6893.549	TwoLocal	COBYLA	Z	pairwise	-
0.088	0.230	19663.618	EfficientSU2	COBYLA	Z	linear	-
0.092	0.232	6403.135	RealAmplitudes	COBYLA	Z	full	-
0.101	0.235	14108.135	EfficientSU2	COBYLA	Z	sca	-
0.108	0.236	6180.985	RealAmplitudes	COBYLA	Z	linear	-
0.119	0.263	5259.155	TwoLocal	COBYLA	Z	circular	-
0.130	0.288	24322.481	PauliTwoDesign	SPSA	Z	-	-
0.135	0.283	31254.089	RealAmplitudes	Nelder-Mead	Z	sca	-
0.156	0.308	7197.127	PauliTwoDesign	COBYLA	Z	-	-
0.172	0.324	22179.464	TwoLocal	SPSA	ZZ	sca	linear
0.175	0.326	19152.350	TwoLocal	SPSA	ZZ	linear	pairwise
0.178	0.333	23610.080	TwoLocal	SPSA	ZZ	circular	linear
0.179	0.339	25874.655	RealAmplitudes	SPSA	ZZ	sca	pairwise
0.182	0.337	14083.206	TwoLocal	SPSA	ZZ	pairwise	linear
0.186	0.339	25461.979	RealAmplitudes	SPSA	ZZ	linear	pairwise
0.188	0.348	31031.320	EfficientSU2	SPSA	ZZ	circular	circular
0.189	0.344	20801.656	TwoLocal	SPSA	ZZ	sca	pairwise
0.189	0.343	19712.909	TwoLocal	SPSA	ZZ	linear	linear
0.189	0.344	25518.044	RealAmplitudes	SPSA	ZZ	linear	linear
0.190	0.346	25699.995	RealAmplitudes	SPSA	ZZ	circular	linear

Table D.1: QML California: Top 40 Configurations

MSE	MAE	Time	Ansatz	Opt.	Feat.	Ent.	Feat.Ent.
0.056	0.179	30269.974	EfficientSU2	COBYLA	Z	full	-
0.057	0.191	26742.804	EfficientSU2	SPSA	Z	sca	-
0.057	0.180	18172.775	TwoLocal	COBYLA	Z	sca	-
0.058	0.185	21107.948	TwoLocal	COBYLA	Z	full	-
0.058	0.182	16235.631	RealAmplitudes	COBYLA	Z	full	-
0.058	0.191	28080.271	TwoLocal	SPSA	Z	full	-
0.059	0.185	20357.743	TwoLocal	COBYLA	Z	circular	-
0.060	0.178	24522.535	RealAmplitudes	SPSA	Z	circular	-
0.071	0.189	15913.069	RealAmplitudes	COBYLA	Z	sca	-
0.075	0.194	24212.491	RealAmplitudes	SPSA	Z	sca	-
0.087	0.211	16152.866	TwoLocal	COBYLA	Z	linear	-
0.087	0.228	26534.667	EfficientSU2	SPSA	Z	linear	-
0.087	0.215	17680.557	RealAmplitudes	COBYLA	Z	linear	-
0.088	0.230	24204.161	TwoLocal	SPSA	Z	linear	-
0.089	0.219	27614.995	RealAmplitudes	SPSA	Z	full	-
0.092	0.221	16184.174	TwoLocal	COBYLA	Z	pairwise	-
0.096	0.224	26393.760	EfficientSU2	COBYLA	Z	sca	-
0.100	0.236	23752.035	TwoLocal	SPSA	Z	pairwise	-
0.103	0.236	14749.277	RealAmplitudes	COBYLA	Z	circular	-
0.104	0.232	26600.699	EfficientSU2	COBYLA	Z	linear	-
0.106	0.230	23996.865	TwoLocal	SPSA	Z	circular	-
0.109	0.245	24124.928	RealAmplitudes	SPSA	Z	linear	-
0.138	0.290	24023.441	TwoLocal	SPSA	Z	sca	-
0.139	0.297	30024.250	EfficientSU2	SPSA	Z	full	-
0.140	0.274	26386.966	EfficientSU2	COBYLA	Z	circular	-
0.164	0.323	26860.893	EfficientSU2	SPSA	Z	circular	-
0.184	0.339	17797.217	RealAmplitudes	COBYLA	ZZ	linear	linear
0.184	0.352	16415.434	RealAmplitudes	COBYLA	ZZ	circular	linear
0.185	0.347	44202.730	TwoLocal	Nelder-Mead	Z	full	-
0.188	0.348	29748.183	PauliTwoDesign	SPSA	Z	-	-
0.190	0.367	41395.693	TwoLocal	Nelder-Mead	Z	pairwise	-
0.190	0.357	19067.599	TwoLocal	SPSA	ZZ	linear	pairwise
0.197	0.368	12713.526	TwoLocal	COBYLA	ZZ	circular	circular
0.197	0.367	28705.283	RealAmplitudes	SPSA	ZZ	sca	linear
0.198	0.369	25422.357	TwoLocal	SPSA	ZZ	linear	sca
0.198	0.372	28507.963	RealAmplitudes	SPSA	ZZ	linear	pairwise
0.199	0.368	31112.246	EfficientSU2	SPSA	ZZ	linear	linear
0.200	0.374	19195.440	RealAmplitudes	COBYLA	ZZ	linear	circular
0.201	0.377	28547.934	RealAmplitudes	SPSA	ZZ	linear	linear
0.202	0.375	31129.704	EfficientSU2	SPSA	ZZ	linear	pairwise

Table D.2: QML California with Noise: Top 40 Configurations

MSE	MAE	Time	Ansatz	Opt.	Feat.	Ent.	Feat.Ent.
0.021	0.107	26993.556	EfficientSU2	SPSA	Z	circular	-
0.022	0.111	27585.848	RealAmplitudes	SPSA	Z	full	-
0.023	0.117	22055.667	TwoLocal	SPSA	Z	sca	-
0.023	0.116	31995.870	EfficientSU2	SPSA	Z	full	-
0.025	0.121	21314.272	TwoLocal	SPSA	Z	linear	-
0.026	0.121	21674.199	RealAmplitudes	SPSA	Z	circular	-
0.026	0.125	15914.194	EfficientSU2	COBYLA	Z	circular	-
0.028	0.121	26686.717	EfficientSU2	SPSA	Z	sca	-
0.028	0.129	21340.496	RealAmplitudes	SPSA	Z	linear	-
0.028	0.128	21426.990	EfficientSU2	COBYLA	Z	sca	-
0.030	0.128	6162.309	TwoLocal	COBYLA	Z	sca	-
0.030	0.136	4981.507	TwoLocal	COBYLA	Z	pairwise	-
0.031	0.126	21641.391	RealAmplitudes	SPSA	Z	sca	-
0.031	0.132	5291.243	RealAmplitudes	COBYLA	Z	circular	-
0.032	0.132	22108.573	TwoLocal	SPSA	Z	circular	-
0.033	0.127	15498.908	EfficientSU2	COBYLA	Z	linear	-
0.033	0.135	21512.028	TwoLocal	SPSA	Z	pairwise	-
0.033	0.131	7890.611	RealAmplitudes	COBYLA	Z	full	-
0.034	0.129	6355.223	TwoLocal	COBYLA	Z	circular	-
0.035	0.140	26007.522	EfficientSU2	SPSA	Z	linear	-
0.040	0.153	13296.613	EfficientSU2	COBYLA	Z	full	-
0.040	0.141	26930.896	TwoLocal	SPSA	Z	full	-
0.040	0.159	5484.585	TwoLocal	COBYLA	Z	full	-
0.041	0.151	7953.380	RealAmplitudes	COBYLA	Z	sca	-
0.042	0.153	25314.528	PauliTwoDesign	SPSA	Z	-	-
0.044	0.155	5669.591	RealAmplitudes	COBYLA	Z	linear	-
0.045	0.155	5645.631	TwoLocal	COBYLA	Z	linear	-
0.057	0.181	6402.417	PauliTwoDesign	COBYLA	Z	-	-
0.058	0.178	24006.525	TwoLocal	SPSA	ZZ	circular	linear
0.061	0.190	31070.289	EfficientSU2	SPSA	ZZ	linear	linear
0.063	0.189	26063.650	RealAmplitudes	SPSA	ZZ	circular	pairwise
0.063	0.184	31400.530	TwoLocal	SPSA	ZZ	full	linear
0.063	0.184	5905.821	TwoLocal	COBYLA	ZZ	circular	linear
0.064	0.191	26130.545	RealAmplitudes	SPSA	ZZ	sca	linear
0.064	0.193	26836.338	RealAmplitudes	SPSA	ZZ	sca	circular
0.064	0.189	26563.177	RealAmplitudes	SPSA	ZZ	circular	linear
0.064	0.189	4967.603	TwoLocal	COBYLA	ZZ	circular	pairwise
0.065	0.196	25824.144	RealAmplitudes	SPSA	ZZ	linear	pairwise
0.065	0.196	48161.706	EfficientSU2	Nelder-Mead	Z	sca	-
0.065	0.195	31105.918	EfficientSU2	SPSA	ZZ	sca	pairwise

Table D.3: QML Seoul: Top 40 Configurations

MSE	MAE	Time	Ansatz	Opt.	Feat.	Ent.	Feat.Ent.
0.022	0.118	24221.327	RealAmplitudes	SPSA	Z	circular	-
0.023	0.117	26612.268	EfficientSU2	SPSA	Z	linear	-
0.027	0.124	23840.443	TwoLocal	SPSA	Z	circular	-
0.028	0.124	26868.753	EfficientSU2	SPSA	Z	circular	-
0.028	0.126	23862.986	TwoLocal	SPSA	Z	sca	-
0.028	0.121	29998.189	EfficientSU2	SPSA	Z	full	-
0.033	0.137	26165.996	EfficientSU2	COBYLA	Z	linear	-
0.034	0.134	13614.698	TwoLocal	COBYLA	Z	sca	-
0.034	0.132	19090.318	RealAmplitudes	COBYLA	Z	sca	-
0.034	0.132	23726.807	TwoLocal	SPSA	Z	linear	-
0.034	0.132	24255.181	RealAmplitudes	SPSA	Z	sca	-
0.035	0.144	16122.550	RealAmplitudes	COBYLA	Z	linear	-
0.036	0.142	23645.806	TwoLocal	SPSA	Z	pairwise	-
0.036	0.137	18323.561	TwoLocal	COBYLA	Z	full	-
0.037	0.147	24022.173	RealAmplitudes	SPSA	Z	linear	-
0.037	0.147	14867.535	TwoLocal	COBYLA	Z	linear	-
0.037	0.141	27906.740	RealAmplitudes	SPSA	Z	full	-
0.037	0.145	26697.718	EfficientSU2	SPSA	Z	sca	-
0.038	0.143	15103.154	RealAmplitudes	COBYLA	Z	circular	-
0.041	0.150	29849.038	EfficientSU2	COBYLA	Z	full	-
0.041	0.141	14220.316	TwoLocal	COBYLA	Z	circular	-
0.045	0.156	14085.185	RealAmplitudes	COBYLA	Z	full	-
0.047	0.160	12117.524	TwoLocal	COBYLA	Z	pairwise	-
0.047	0.159	21641.521	EfficientSU2	COBYLA	Z	circular	-
0.050	0.163	27207.133	TwoLocal	SPSA	Z	full	-
0.054	0.178	17493.903	PauliTwoDesign	COBYLA	Z	-	-
0.056	0.172	21026.555	EfficientSU2	COBYLA	Z	sca	-
0.056	0.166	29622.878	PauliTwoDesign	SPSA	Z	-	-
0.058	0.174	38422.478	RealAmplitudes	Nelder-Mead	Z	linear	-
0.060	0.177	29338.381	RealAmplitudes	SPSA	ZZ	sca	circular
0.061	0.178	28764.398	RealAmplitudes	SPSA	ZZ	circular	linear
0.062	0.183	31140.786	EfficientSU2	SPSA	ZZ	linear	linear
0.062	0.181	24335.840	TwoLocal	SPSA	ZZ	circular	pairwise
0.062	0.177	35029.198	EfficientSU2	SPSA	ZZ	full	pairwise
0.062	0.181	28877.776	RealAmplitudes	SPSA	ZZ	sca	linear
0.063	0.176	32311.150	RealAmplitudes	SPSA	ZZ	full	linear
0.063	0.186	50972.894	EfficientSU2	Nelder-Mead	Z	sca	-
0.063	0.167	44613.772	RealAmplitudes	Nelder-Mead	Z	full	-
0.063	0.186	32356.753	TwoLocal	SPSA	ZZ	full	linear
0.064	0.182	51450.466	EfficientSU2	Nelder-Mead	Z	circular	-

Table D.4: QML Seoul With Noise: Top 40 Configurations

Acc.	F-1	Time	Ansatz	Opt.	Feat.	Ent.	Feat.Ent.	Prep.
0.588	0.583	606	TwoLocal	COBYLA	Z	pairwise	-	LDA
0.576	0.576	3171	EfficientSU2	SPSA	Z	sca	-	LDA
0.572	0.569	3208	EfficientSU2	SPSA	Z	circular	-	LDA
0.584	0.564	1035	TwoLocal	COBYLA	Z	circular	-	LDA
0.588	0.564	3185	EfficientSU2	SPSA	Z	linear	-	LDA
0.568	0.554	2640	RealAmplitudes	SPSA	Z	circular	-	LDA
0.540	0.548	926	TwoLocal	COBYLA	Z	full	-	LDA
0.548	0.547	2639	TwoLocal	SPSA	Z	circular	-	LDA
0.560	0.546	2615	TwoLocal	SPSA	Z	sca	-	LDA
0.548	0.540	2598	RealAmplitudes	SPSA	Z	linear	-	LDA
0.540	0.536	2578	TwoLocal	SPSA	Z	pairwise	-	LDA
0.540	0.534	2570	TwoLocal	SPSA	Z	linear	-	LDA
0.544	0.533	1001	RealAmplitudes	COBYLA	Z	circular	-	LDA
0.532	0.532	3579	EfficientSU2	SPSA	Z	full	-	LDA
0.528	0.525	2478	EfficientSU2	COBYLA	Z	circular	-	LDA
0.540	0.519	1865	EfficientSU2	COBYLA	Z	sca	-	LDA
0.520	0.517	2889	TwoLocal	SPSA	Z	full	-	LDA
0.528	0.515	2295	EfficientSU2	COBYLA	Z	linear	-	LDA
0.524	0.512	2952	RealAmplitudes	SPSA	Z	full	-	LDA
0.496	0.507	422	TwoLocal	COBYLA	Z	linear	-	LDA
0.492	0.504	674	RealAmplitudes	COBYLA	Z	sca	-	LDA
0.480	0.488	2661	RealAmplitudes	SPSA	Z	sca	-	LDA
0.496	0.484	1132	RealAmplitudes	COBYLA	Z	linear	-	LDA
0.464	0.483	989	RealAmplitudes	COBYLA	Z	full	-	LDA
0.444	0.466	301	TwoLocal	COBYLA	Z	sca	-	LDA
0.400	0.454	2634	PauliTwoDesign	SPSA	Z	-	-	LDA
0.472	0.453	916	RealAmplitudes	COBYLA	Z	circular	-	PCA
0.456	0.441	2201	EfficientSU2	COBYLA	Z	full	-	LDA
0.444	0.440	3143	RealAmplitudes	SPSA	Z	sca	-	PCA
0.436	0.426	3783	EfficientSU2	SPSA	Z	linear	-	PCA
0.404	0.423	642	TwoLocal	COBYLA	Z	linear	-	PCA
0.412	0.423	733	RealAmplitudes	COBYLA	Z	sca	-	PCA
0.416	0.421	2218	EfficientSU2	COBYLA	Z	linear	-	PCA
0.412	0.412	3586	RealAmplitudes	SPSA	Z	full	-	PCA
0.404	0.402	3805	EfficientSU2	SPSA	Z	sca	-	PCA
0.388	0.400	2420	EfficientSU2	COBYLA	Z	full	-	PCA
0.420	0.398	882	TwoLocal	COBYLA	Z	full	-	PCA
0.416	0.397	3044	TwoLocal	SPSA	Z	linear	-	PCA
0.436	0.395	4292	EfficientSU2	SPSA	Z	full	-	PCA
0.392	0.394	3092	RealAmplitudes	SPSA	Z	linear	-	PCA

Table D.5: QML Cover Type: Top 40 Configurations

Acc.	F-1	Time	Ansatz	Opt.	Feat.	Ent.	Feat.Ent.	Prep.
0.560	0.551	2230	RealAmplitudes	COBYLA	Z	circular	-	LDA
0.564	0.550	1730	TwoLocal	COBYLA	Z	pairwise	-	LDA
0.544	0.536	3016	RealAmplitudes	SPSA	Z	linear	-	LDA
0.528	0.532	3043	RealAmplitudes	SPSA	Z	circular	-	LDA
0.532	0.528	2536	TwoLocal	COBYLA	Z	sca	-	LDA
0.548	0.527	3362	EfficientSU2	COBYLA	Z	circular	-	LDA
0.540	0.526	3328	EfficientSU2	COBYLA	Z	linear	-	LDA
0.532	0.525	3007	TwoLocal	SPSA	Z	pairwise	-	LDA
0.528	0.519	1856	TwoLocal	COBYLA	Z	circular	-	LDA
0.536	0.518	3023	TwoLocal	SPSA	Z	linear	-	LDA
0.528	0.517	2165	RealAmplitudes	COBYLA	Z	sca	-	LDA
0.504	0.514	1682	TwoLocal	COBYLA	Z	linear	-	LDA
0.516	0.510	3438	TwoLocal	SPSA	Z	full	-	LDA
0.512	0.507	3389	EfficientSU2	SPSA	Z	circular	-	LDA
0.512	0.505	3748	EfficientSU2	SPSA	Z	full	-	LDA
0.520	0.499	3322	EfficientSU2	COBYLA	Z	sca	-	LDA
0.492	0.497	3696	EfficientSU2	COBYLA	Z	full	-	LDA
0.508	0.494	3028	TwoLocal	SPSA	Z	circular	-	LDA
0.496	0.490	2260	RealAmplitudes	COBYLA	Z	linear	-	LDA
0.492	0.481	2282	RealAmplitudes	COBYLA	Z	full	-	LDA
0.452	0.480	3047	RealAmplitudes	SPSA	Z	sca	-	LDA
0.484	0.474	5568	RealAmplitudes	COBYLA	Z	linear	-	PCA
0.424	0.471	3053	TwoLocal	SPSA	Z	sca	-	LDA
0.444	0.467	2481	TwoLocal	COBYLA	Z	full	-	LDA
0.468	0.466	3393	EfficientSU2	SPSA	Z	linear	-	LDA
0.464	0.462	3562	PauliTwoDesign	SPSA	Z	-	-	LDA
0.444	0.442	3387	EfficientSU2	SPSA	Z	sca	-	LDA
0.436	0.437	8864	TwoLocal	SPSA	Z	full	-	PCA
0.416	0.434	8860	RealAmplitudes	SPSA	Z	full	-	PCA
0.444	0.431	7686	TwoLocal	SPSA	Z	linear	-	PCA
0.420	0.425	3387	RealAmplitudes	SPSA	Z	full	-	LDA
0.428	0.424	4318	TwoLocal	COBYLA	Z	linear	-	PCA
0.440	0.417	6506	RealAmplitudes	COBYLA	Z	full	-	PCA
0.400	0.416	6240	TwoLocal	COBYLA	Z	circular	-	PCA
0.432	0.413	7736	RealAmplitudes	SPSA	Z	circular	-	PCA
0.432	0.408	6160	RealAmplitudes	COBYLA	Z	circular	-	PCA
0.368	0.404	3795	TwoLocal	COBYLA	Z	pairwise	-	PCA
0.356	0.397	9624	PauliTwoDesign	SPSA	Z	-	-	PCA
0.380	0.391	9730	EfficientSU2	SPSA	Z	full	-	PCA
0.388	0.388	7723	RealAmplitudes	SPSA	Z	sca	-	PCA

Table D.6: QML Cover Type with Noise: Top 40 Configurations

Acc.	F-1	Time	Ansatz	Opt.	Feat.	Ent.	Feat.Ent.	Prep.
0.912	0.905	2016	TwoLocal	SPSA	Z	full	-	PCA
0.904	0.898	2906	EfficientSU2	COBYLA	Z	circular	-	PCA
0.884	0.890	2576	RealAmplitudes	SPSA	Z	sca	-	PCA
0.888	0.886	2612	RealAmplitudes	SPSA	Z	circular	-	PCA
0.880	0.886	2457	EfficientSU2	COBYLA	Z	sca	-	PCA
0.860	0.876	2969	RealAmplitudes	SPSA	Z	full	-	PCA
0.872	0.872	2144	EfficientSU2	COBYLA	Z	linear	-	PCA
0.864	0.869	847	TwoLocal	COBYLA	Z	sca	-	PCA
0.860	0.865	1385	TwoLocal	COBYLA	Z	full	-	PCA
0.824	0.847	698	RealAmplitudes	COBYLA	Z	sca	-	PCA
0.840	0.842	3292	EfficientSU2	COBYLA	Z	full	-	PCA
0.824	0.841	1144	RealAmplitudes	COBYLA	Z	circular	-	PCA
0.852	0.839	2589	RealAmplitudes	SPSA	Z	linear	-	PCA
0.844	0.836	993	TwoLocal	COBYLA	Z	linear	-	PCA
0.832	0.833	3242	EfficientSU2	SPSA	Z	linear	-	PCA
0.824	0.831	3603	EfficientSU2	SPSA	Z	full	-	PCA
0.824	0.826	1162	RealAmplitudes	COBYLA	Z	full	-	PCA
0.840	0.825	973	RealAmplitudes	COBYLA	Z	linear	-	PCA
0.800	0.817	922	TwoLocal	COBYLA	ZZ	circular	pairwise	PCA
0.820	0.816	803	TwoLocal	COBYLA	Z	pairwise	-	PCA
0.828	0.810	1732	TwoLocal	SPSA	Z	circular	-	PCA
0.792	0.803	1401	RealAmplitudes	COBYLA	ZZ	circular	pairwise	PCA
0.784	0.798	3641	EfficientSU2	COBYLA	ZZ	full	linear	PCA
0.796	0.798	1198	RealAmplitudes	COBYLA	ZZ	circular	sca	PCA
0.820	0.797	3256	EfficientSU2	SPSA	Z	circular	-	PCA
0.804	0.794	1314	TwoLocal	COBYLA	Z	circular	-	PCA
0.760	0.789	3475	EfficientSU2	COBYLA	ZZ	sca	linear	PCA
0.812	0.787	3200	EfficientSU2	SPSA	Z	sca	-	PCA
0.760	0.786	1491	TwoLocal	Nelder-Mead	Z	sca	-	PCA
0.760	0.777	3212	RealAmplitudes	SPSA	ZZ	circular	circular	PCA
0.756	0.776	1244	TwoLocal	COBYLA	ZZ	full	sca	PCA
0.736	0.776	3456	EfficientSU2	COBYLA	ZZ	circular	sca	PCA
0.744	0.776	3753	EfficientSU2	SPSA	ZZ	sca	linear	PCA
0.780	0.775	3318	EfficientSU2	COBYLA	ZZ	linear	pairwise	PCA
0.824	0.775	1707	TwoLocal	SPSA	Z	linear	-	PCA
0.756	0.771	3610	RealAmplitudes	SPSA	ZZ	full	pairwise	PCA
0.812	0.768	1690	TwoLocal	SPSA	Z	pairwise	-	PCA
0.768	0.767	1554	RealAmplitudes	COBYLA	ZZ	sca	linear	PCA
0.760	0.767	3445	EfficientSU2	COBYLA	ZZ	circular	linear	PCA
0.780	0.766	3418	EfficientSU2	COBYLA	ZZ	sca	pairwise	PCA

Table D.7: QML KDD: Top 40 Configurations

Acc.	F-1	Time	Ansatz	Opt.	Feat.	Ent.	Feat.Ent.	Prep.
0.864	0.859	7677	TwoLocal	COBYLA	Z	circular	-	PCA
0.860	0.854	10241	EfficientSU2	SPSA	Z	circular	-	PCA
0.860	0.852	9142	TwoLocal	SPSA	Z	linear	-	PCA
0.836	0.846	8504	EfficientSU2	COBYLA	Z	sca	-	PCA
0.848	0.840	9245	RealAmplitudes	SPSA	Z	circular	-	PCA
0.820	0.833	9259	TwoLocal	SPSA	Z	sca	-	PCA
0.836	0.832	6659	RealAmplitudes	COBYLA	Z	sca	-	PCA
0.828	0.808	7895	TwoLocal	COBYLA	Z	full	-	PCA
0.804	0.805	11634	EfficientSU2	SPSA	Z	full	-	PCA
0.788	0.799	8789	RealAmplitudes	COBYLA	Z	full	-	PCA
0.816	0.793	7467	TwoLocal	COBYLA	Z	sca	-	PCA
0.776	0.792	10209	RealAmplitudes	COBYLA	ZZ	sca	circular	PCA
0.816	0.790	8525	EfficientSU2	COBYLA	Z	circular	-	PCA
0.756	0.788	11094	EfficientSU2	COBYLA	ZZ	linear	sca	PCA
0.776	0.788	13282	EfficientSU2	SPSA	ZZ	sca	circular	PCA
0.772	0.787	10558	TwoLocal	SPSA	ZZ	circular	sca	PCA
0.808	0.782	8580	EfficientSU2	COBYLA	Z	linear	-	PCA
0.760	0.782	6779	TwoLocal	COBYLA	ZZ	sca	sca	PCA
0.756	0.780	11049	EfficientSU2	COBYLA	ZZ	linear	circular	PCA
0.764	0.778	9368	TwoLocal	COBYLA	ZZ	full	linear	PCA
0.796	0.778	7084	TwoLocal	COBYLA	Z	linear	-	PCA
0.768	0.778	3526	TwoLocal	COBYLA	ZZ	pairwise	circular	PCA
0.756	0.776	6083	RealAmplitudes	COBYLA	Z	circular	-	PCA
0.812	0.771	9242	TwoLocal	SPSA	Z	pairwise	-	PCA
0.756	0.766	12150	RealAmplitudes	SPSA	ZZ	sca	linear	PCA
0.816	0.765	10232	EfficientSU2	SPSA	Z	linear	-	PCA
0.816	0.765	5906	RealAmplitudes	COBYLA	Z	linear	-	PCA
0.752	0.761	10223	RealAmplitudes	COBYLA	ZZ	circular	circular	PCA
0.784	0.757	9214	RealAmplitudes	SPSA	Z	linear	-	PCA
0.732	0.756	12256	RealAmplitudes	SPSA	ZZ	linear	sca	PCA
0.724	0.755	12281	RealAmplitudes	SPSA	ZZ	circular	circular	PCA
0.724	0.755	14716	EfficientSU2	SPSA	ZZ	full	circular	PCA
0.756	0.755	14586	EfficientSU2	SPSA	ZZ	full	pairwise	PCA
0.744	0.754	11084	EfficientSU2	COBYLA	ZZ	sca	circular	PCA
0.728	0.753	12265	RealAmplitudes	SPSA	ZZ	linear	circular	PCA
0.736	0.748	13350	EfficientSU2	SPSA	ZZ	linear	sca	PCA
0.768	0.747	10278	EfficientSU2	SPSA	Z	circular	-	LDA
0.720	0.747	9151	TwoLocal	COBYLA	ZZ	full	sca	PCA
0.756	0.746	10614	RealAmplitudes	COBYLA	ZZ	full	pairwise	PCA
0.744	0.745	7353	TwoLocal	COBYLA	ZZ	pairwise	pairwise	PCA

Table D.8: QML KDD with Noise: Top 40 Configurations