Degree Project in Computer Science and Engineering

First cycle 15 credits

An Evaluation of Classical and Quantum Kernels for Machine Learning Classifiers

KTH Bachelor Thesis Report

JACOB WESTERGREN AND TEO NORDSTRÖM

Stockholm, Sweden 2023

Degree Project in Computer Science and Engineering, First Cycle 15 credits
Date: June 7, 2023
Supervisor: Stefano Markidis
Examiner: Pawel Herman
Swedish title: En utvärdering av klassiska och kvantkärnor inom maskininlärnings klassifikationsmodeller
School of Electrical Engineering and Computer Science

Abstract

Quantum computing is an emerging field with potential applications in machine learning. This research project aimed to compare the performance of a quantum kernel to that of a classical kernel in machine learning binary classification tasks. Two Support Vector Machines, a popular classification model, was implemented for the respective Variational Quantum kernel and the classical Radial Basis Function kernel and tested on the same sets of artificial quantum-based testing data. The results show that the quantum kernel significantly outperformed the classical kernel for the specific type of data and parameters used in the study. The findings suggest that quantum kernels have the potential to improve machine learning performance for certain types of problems, such as search engines and self-driving vehicles. Further research is, however, needed to confirm their utility in general situations.

Keywords

Machine Learning, Quantum Computing, Kernels, Support Vector Machines

Sammanfattning

Kvantberäkning är ett växande forskningsområde med möjliga tillämpningar inom maskininlärning. I detta forskningsprojekt jämfördes prestandan hos en klassisk kärna med den hos en kvantkärna i binär klassificering för maskininlärninguppgifter, och implikationerna av resultaten diskuterades. Genom att implementera två stödvektormaskiner, en populär klassifikationsmodell, för respektive variabel kvantkärna och klassisk radiell basfunktionskärna kunde vi direkt testa båda kärnorna på samma uppsättning av artificiella kvant-baserad testdata. Resultaten visar på betydande prestandafördelar för kvantkärnan jämfört med den klassiska kärnan när det gäller denna specifika typ av data och de parametrar som användes i vår studie. Vi drar slutsatsen att kvantkärnor inom maskininlärning har potential att överträffa klassiska kärnor, men att mer forskning krävs för att fastställa om detta har någon nytta i allmänna situationer. Om det finns betydande prestandafördelar kan det finnas många tillämpningar, till exempel för sökmotorer och självkörande fordon.

Nyckelord

Maskininlärning, Kvantberäkning, Kärnor, Stödvektormaskin

Acknowledgements

Thank you to our supervisor Stefano Markidis for being great to work with, and for holding biweekly meetings where we could voice any questions or concerns we had about our paper. We would also like to thank Pawel Herman, the examinator, for the ability to write and release this paper.

Authors

Jacob Westergren jawester@kth.se
Teo Nordström teon@kth.se
Computer Science
KTH Royal Institute of Technology

Place for Project

Stockholm, Sweden
KTH Campus Valhallavägen

Examiner

Pawel Hermam
Division of Computational Science and Technology
KTH Royal Institute of Technology

Supervisor

Stefano Markidis
Division of Computational Science and Technology
KTH Royal Institute of Technology

Abbreviations

SVM	Support Vector Machine
RBF	Radial Basis Function
VQK	Variational Quantum Kernel
ROC	Receiver Operating Characteristic
AUC	Area Under Curve

Contents

1 Introduction

In recent years, the field of quantum computing has seen a lot of progress. With the active development of powerful quantum computers, there has been growing interest in finding suitable applications for the possibilities that quantum computing may allow. Some powerful quantum computing algorithms have existed for decades, such as Shor's Algorithm (1994), which can be used to quickly (relative to classical computers) find prime factors of large integers [1]. While never fully implemented, algorithms like Shor's Algorithm has spurred interest and development within the area of quantum computing. This rise in interest has also read to more applications being theorised and developed. One of the interest areas which has become more popular in later years has been quantum computing's applications within machine learning.

Machine learning is an old field, stretching back to the early days of computation. The idea that a machine potentially could learn and think is both interesting and would have many beneficial applications. Already being used for such things as search engines, personal recommendations, voice recognition and self driving vehicles, machine learning has a lot of applications that humans today can only theorise about. Within the area of machine learning, quantum computers have been theorised to potentially outperform normal computers under certain circumstances.

One specific application where quantum computers could potentially assist in machine learning would be in its use of kernels for classifying data. Within machine learning, classifying data is important since it allows the computer to discern between different types of data through their inherent attributes. The kernel helps in this by efficiently calculating the similarity or distance between datapoints in a transformed feature space. With the help of quantum computing, it is thought that the calculation ability of these kernels could become even more effective [2].

1.1 Research Question

The study's purpose is to investigate the potential advantages of quantum kernels over classical kernels in machine learning classifiers. Specifically, the study seeks to answer the following questions:

- How does the performance of a classical kernel compare to quantum kernel classifiers in binary classification tasks?
- What are the implications of the results for the potential use of quantum computing in machine learning?

1.2 Scope

The study was focused on binary classification and the kernel classifiers was restricted to two implementations of support vector machines (SVM): a classical SVM using a Radial Basis Function (RBF) kernel and a Quantum-Classical SVM hybrid with a Variational Quantum Kernel. These kernels were chosen due to their importance in their respective fields, as the RBF kernel is a widely used kernel in classical SVMs and the VQK is one of the most promising variant of quantum kernels. Therefore, a comparison between the established RBF kernel and the emerging VQK should provide valuable insights.

The experiments' datasets were all locally generated synthetic datasets. These were generated using simulated quantum circuits to produce quantum-based data that would be fully linearly separable in the quantum feature space. The chosen parameters for the performance evaluation were: accuracy, precision, recall, F1-score and AUC, based on information in [3]. The results of this comparison provide a deeper understanding of the advantages and disadvantages of classical and quantum kernels in binary classification tasks.

1.3 Relevance

As described in the intro, machine learning is already quite a developed field. Over time, it has gotten integrated into many of the services people use every day, such as search engines. By combining this technology with quantum computing, we aim to see if there is any potential benefits within this intersection of technology.

These benefits could potentially be used to create more powerful machine learning algorithms and lead to more efficient machine learning systems.

1.4 Approach

Two SVMs will be implemented using the SVM class from the Scikit-learn software library [4] and trained on datasets generated by the Qiskit software libraries [5] `ad_hoc_data function`, split into training and test sets. One SVM will use a classical RBF kernel while the other will use a VQK, both of which will be optimized for each classification task using the training datasets. The trained SVMs will then be used to classify the test datasets and the results of these classification tasks will be used as the foundation for the comparison.

1.5 Delimitations

This study had several limitations that needed to be considered when interpreting the results. Firstly, the datasets were all synthetic, meaning they were all generated locally, to guarantee that all the datasets were fully separable by the quantum feature map. The study's results might therefore not be directly comparable to real-world data.

Secondly, the result could not measure the time usage of the different algorithms. This is since quantum algorithms naturally take far longer on simulated hardware compared to actual quantum hardware. What this means is that measuring the time usage would only matter when quantum kernels are simulated which is not a viable or expected use case.

Finally, the study did not utilize any advanced preprocessing on the study's data, which is a technique that can potentially improve the overall quality of a classification model's results. While this may have expedited the implementation of the classification model, it is important to note that a more advanced preprocessing stage may have led to a different outcome than ours.

1.6 Outline

This paper is split into six sections: 1 Introduction, 2 Background, 3 Method, 4 Results, 5 Discussion and 6 Conclusions. The introduction , as its name suggests, introduces the reader to the project and states the research questions. The Background section explains the fundamentals of the research area for the reader and mentions some relevant research that inspired the project. The Method section explains the approach used to gather the necessary results for the research questions, and includes explanations of the chosen evaluation metrics. The Results section illustrates the results from the method using tables and graphs and the Discussion contains the interpretations of these results, what implications they have, and how this research can be built upon. The final section, Conclusion, summarizes the entire paper and ends with some final words. Following this are appendices, which includes data and figures that are important but did not fit in the paper.

2 Background

This section provides an overview of the key concepts and theories relevant to the study's research. The focus is on machine learning, specifically the fundamental principles behind SVMs and kernels, as well as quantum computing and how it can be implemented into the mentioned classification algorithms. Additionally, it also contains a literature review subsection of similar research studies, detailing their usage for this study's research purposes.

2.1 Machine Learning

Machine learning is a rising subfield in statistics focused on the systematic study of algorithms that can learn and improve from experience without being explicitly programmed. This field primarily involves two types of learning experiences: supervised and unsupervised learning. In supervised learning, the algorithm is given full information about each datapoint's attributes and its associated response, while in unsupervised learning, the algorithm is given only the attributes. The goal of both of these approaches is to create a model that can accurately predict the response for new, unseen datapoints. Depending on the type of the response variable, the algorithm can belong to one of two groups: regression problems, which involve predicting a continuous response, or classification problems, which involve predicting a categorical response [3].

There are a wide variety of statistical learning algorithms used in machine learning, such as support vector machines and neural networks, and selecting the best approach for a given problem is vital [6]. The optimal approach can be found by testing different models on the data and using evaluation techniques, such as cross-validation and hyperparameter tuning, to construct a comparison between the models and the best suited model can thus be found.

2.2 Support Vector Machines

Support vector machines are a widely used machine learning technique that can be used for binary classification tasks. For classification, the fundamental concept behind SVMs is to separate datapoints from the input space $\{(\vec{x}_i, y_i) : \vec{x}_i \in \mathbb{R}^N, y_i =$

$\pm 1\}_{(i=1...M)}$ into two half-spaces, one for each class, using a hyperplane [7]. Of all the possible separating hyperplanes, the maximal margin hyperplane is chosen, which refers to the separating hyperplane with the biggest margin. The margin, which is defined as the distance between the nearest point in each class to the hyperplane in the input space, can be expressed as $\frac{2}{|\vec{w}|}$ where $\vec{w}$ is the hyperplane's normal vector [8]. Therefore the optimal hyperplane can also be expressed as the hyperplane with the smallest magnitude of $\vec{w}$, subject to the inequality constraint $y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1$, where b is the bias term, meaning that no misclassifications on the input data are allowed.

However, many datasets are non-separable and thus many SVMs utilize a 'soft-margin', which adds positive slack variables ξ_i $(i = 1, ..., M)$, one for each datapoint, that relaxes the inequality constraint. To control the trade-off between maximizing the margin for correctly classified points and allowing for some misclassifications, a regularization parameter C is introduced. This parameter limits the sum of all slack variables $\sum_i^M \xi_i$ and can thus be used to control the tolerated error levels. The final hyperplane is obtained as the solution to this minimization problem, subject to the constraint $y_i(\vec{x}_i \cdot \vec{w} + b) \geq 1 - \xi_i$ and $\xi_i \geq 0$:

$$L_P(\vec{w}, \xi, b, \Lambda, R) = \frac{1}{2}|\vec{w}|^2 + C\sum_i \xi_i - \sum_{i=i}^{M} \alpha_i(y_i(\vec{w} \cdot \vec{x}_i + b) - 1 + \xi_i) - \sum_i u_i\xi_i \quad (1)$$

where R is the vector of the Lagrange multipliers used to enforce the positivity of the slack variables ξ_i [9]. This primal formulation L_P is quite complex; Therefore, the dual maximization problem (L_D) is commonly solved in SVMs instead. The dual formulation relies solely on the constraints imposed by the Lagrange multipliers used for the margin enforcement and the inner products between the datapoint pairs [10]. The dual is thus easier to solve and has the following formulation:

$$L_D(\Lambda) = \sum_{i=1}^{M} \alpha_i - \frac{1}{2}\sum_{i,j=1}^{M} \alpha_i\alpha_j y_i y_j \vec{x}_i \cdot \vec{x}_j \quad (2)$$

subject to the constraints $\sum_{i=1}^{M} \alpha_i y_i = 0$ and $0 \leq \alpha_i \leq C$ [10],

2.2.1 Non-linear decision boundary

However, there are cases where the decision boundary is not linear, and thus a separating hyperplane in the input space cannot be found. To overcome this limitation, SVMs use a high-dimensional map Φ to transform all the datapoints to a high-dimensional feature space, allowing the separation of non-linear, non-separable data in the input space, as demonstrated in 2.1. However, calculating the inner product of feature vectors in a high-dimensional feature space can be computationally expensive for high-dimensional data sets [11]. To address this issue, SVMs use kernels $k(\vec{x}_i, \vec{x}_j)$, functions that calculates the inner product between a pair of datapoints $\vec{x}_i$ and $\vec{x}_j$ in the feature space without actually transforming them to the feature space [11]. There are various types of kernels that are suitable for different data sets, and more information on kernels can be found in Section 2.3. With the implementation of kernels, the final formulation of the dual optimization problem is as follows, subject to the same constraints as before [10]:

$$L_D(\Lambda) = \sum_{i=1}^{M} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{M} \alpha_i \alpha_j y_i y_j k(\vec{x}_i, \vec{x}_j) \tag{3}$$

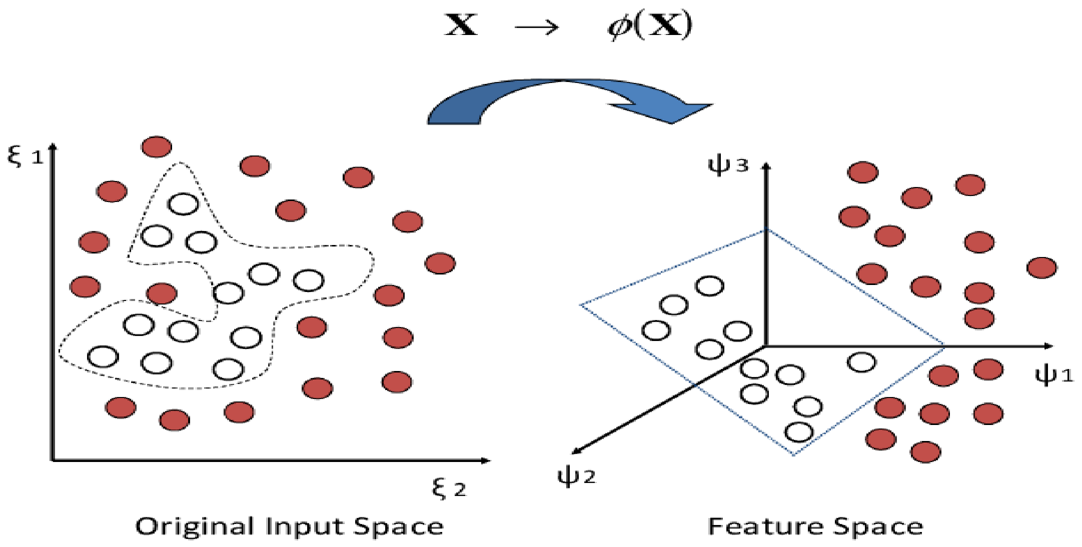


Figure 2.1: The figure shows how applying a map ϕ to transform the datapoints to a feature space enables the separation of non-separable data in the input space. Image sourced from [12].

2.2.2 Hyperplane Construction and Usage

A common algorithm used to solve the dual optimization problem described by Eq. (3) is the Sequential Minimal Optimization, which iteratively solves the problem by solving two Lagrange multipliers at a time. After solving the dual optimization problem, the parameters of the hyperplane can be computed using only the datapoints with non-zero Lagrangian multipliers, known as 'support vectors'. These support vectors are datapoints located either on the margin or on the wrong side of it [7]. The support vectors only make up a small subset of all datapoints, and therefore the computation of $\vec{w}$ and b is very efficient. These variables can then be used to construct the maximal margin hyperplane, which can be used to classify new, unseen datapoints using the indicator function:

$$y(\vec{u}) = sign(\sum_{i=1}^{N_s} \alpha_i y_i k(\vec{s}_i, \vec{u}) + b) \tag{4}$$

where s_i are the support vectors and $\vec{u}$ is an unseen datapoint [10]. If the new point is above the hyperplane, its predicted class is $+1$; Otherwise, the new point's predicted class is -1, as illustrated in fig 2.2.

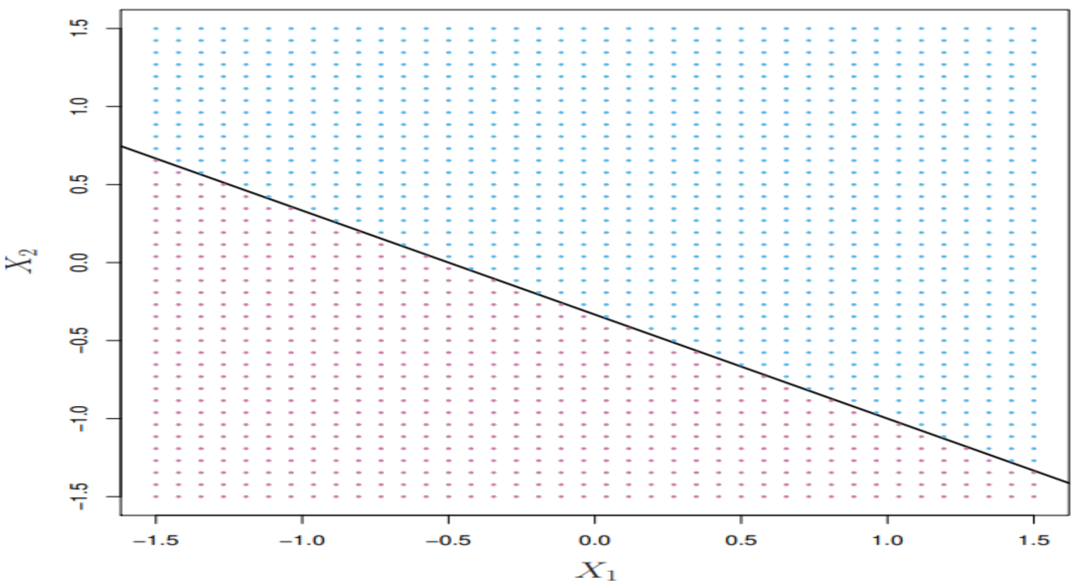


Figure 2.2: The figure demonstrates how the SVM predicts new, unseen datapoints located in the blue area as $+1$, and those in the purple area as -1. Image sourced from [13].

2.3 Kernels

Kernels are positive definite functions that can be expanded in a uniformly convergent series of eigenfunctions ψ_i and eigenvalues λ_i. This expansion is defined as $k(x,y) = \sum_i^{\infty} \lambda_i \psi_i(x)\psi_i(y)$, which correspond to a dot product $(\phi(x_i) \cdot \phi(x_j)$ in the range of $\phi : x \to (\sqrt{\lambda_1}\psi_1(x), \sqrt{\lambda_2}\psi_2(x), ...)$ where ϕ is a mapping to a high-dimensional feature space [14]. Mercer's conditions specify the necessary properties that a kernel must have for this expansion to be valid, and kernels that satisfy these conditions are called Mercer's kernels [10]. The eigenfunctions ψ_i are the basis functions of the reproducing kernel Hilbert space (RKHS) and are dependant on the utilized Mercer kernel. In this space, all necessary computations for the inner product takes place [11]. The Mercer's kernels' ability to compute the inner product of the datapoints pairs in the RKHS, thus circumventing the need to know and use the high-dimensional mapping ϕ, is called the 'kernel trick', and its usage can quicken the computation process.

There are various types of Mercer's kernels, such as polynomial kernels, linear kernels, etc. The one used in this study is a radial basis function (RBF) kernel, which is a Mercer kernel that utilizes an infinite-dimensional feature map. The RBF kernel is defined as $k(x,y) = e^{\frac{-||x-y||^2}{2\sigma^2}}$ [10] and computes the exponential of the negative squared Euclidean distance between two datapoints. The kernel coefficient σ controls the smoothness of the decision boundary, with larger values resulting in smoother and more generalized decision boundaries. By satisfying Mercer's conditions, even though the data is mapped to an infinite-dimensional feature space, the kernel can use the kernel trick and efficiently compute the inner products of all the datapoint pairs in the RKHS space, spanned by the unique basis functions of the RBF kernel. The use of a RBF kernel in a SVM results in a non-linear decision boundary of the form $f(\overrightarrow{u}) = sign(\sum_{i=1}^{M} \alpha_i exp(\frac{|\overrightarrow{u} - \overrightarrow{x_i}|^2}{\sigma^2}))$, where u is an unseen, unlabeled datapoint [9].

2.4 Quantum Computing

Quantum computing is a field of computing that is used to harness the abilities of quantum mechanics to make computing certain kinds of problems faster and more efficient than classical computers[15]. Classical computers, the ones most commonly seen in this day and age, use bits to calculate whatever the user wants the computer to calculate. These bits can only be 0 or 1, off or on, at any specific time. This means the result of the calculations are deterministic but can also be slow since only one possibility can be calculated at a time. Quantum computers instead use physical phenomena and quantum mechanics to allow for some computations to be performed much more efficiently than in classical computers[15].

2.4.1 Quantum Bits

A quantum bit (commonly known as a qubit) is, similar to a bit in classical computing, the fundamental building block of quantum computing [15]. The possible states for a qubit is $|0\rangle$ and $|1\rangle$ which correspond to the 0 and 1 states of a classical bit. However, instead of having a state of either 0 or 1 at any time, qubits can be in many different state at once. This is due to a phenomenon called superposition.

Superposition is a property of microscopically small objects to be in several states at the same time [15]. Instead of existing in one place or another, it is said to be in a "Superposition". Superposition also applies to other properties of an object, meaning it is not limited to only spatial position. The object is, however, only in Superposition when it is not being observed. When one attempts to measure the state of a superposition, this superposition collapses into one singular state. [15].

Through superposition, the qubit can form linear combinations of states:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \tag{5}$$

where α and β are complex numbers [16]. When measured, however, only a state of $|0\rangle$ and $|1\rangle$ can be found since the superposition collapses and the quantum state only provides severely restricted information. This means the measurements of qubits are inherently probabilistic, since only the probability of the superposition collapsing into a specific state could be considered without measuring it [2]. The probabilities of either $|0\rangle$ or $|1\rangle$ being measured would be based on the α and β in the linear combination, with the probability of $|0\rangle$ being measured being $|\alpha|^2$ and $|1\rangle$ being measured being $|\beta|^2$ where $|\alpha|^2 + |\beta|^2 = 1$. Because of this, a qubit can be represented as a vector of length one in a two dimensional complex vector space, a Hilbert space, meaning a single qubit can store a theoretically infinite amount of information. This is in stark contrast to the either 1 or 0 a classical bit can contain. Both kinds of bits will only ever be measured as 1 ($|1\rangle$) or 0 ($|0\rangle$), though. The ability to harness these qubits and their superposition is what gives quantum computing its edge over classical computers in certain regards[16].

Qubits work both in singular and in multiples, with each extra qubit allowing for an exponential amount of extra information to be held. For each extra qubit, the amount of states is doubled, e.g. the total amount of states is 2^n where n is the number of qubits. For each extra qubit, the linear combination is extended by two more variables. The total probability equation for a two qubit system would thus be $|\alpha|^2 + |\beta|^2 + |\gamma|^2 + |\delta|^2 = 1$. When working with pairs of qubits, a commonly utilized state is the “Bell state” which means two linked qubits will always have the same state if one of them is measured. This is a form of quantum entanglement and can be used to transfer information over theoretically infinite distances [16].

2.4.2 Quantum Gates

Quantum gates are the quantum analogue to logic gates and allows for the manipulation of qubits. Since qubits are a linear combination of states instead of a distinct state, it is not possible to directly modify the end state, because all quantum mechanics follow linear behaviour. Instead, gates are utilized to manipulate the linear combinations through transformations (often visualized through matrices). Since the sum of the probability of all outcomes of the measurement of a system of qubits must be 1, all transformation of this linear combination must be unitary. Due to this fact, all quantum gates are also reversible (which means that one always has the ability to know the input if one know the output and the transformation). Outside of this, there are no known limitations on the transformation, which allows for powerful manipulation of one or several qubits [16].

2.4.3 Computation of quantum information

Quantum computation is the act of calculating information using quantum computers. The foundation of these quantum computers are quantum circuits, which can with the help of quantum gates change the quantum states in qubits and harness this information for computational goals. These circuits can be designed to perform quantum algorithms that compute information in a classical or a novel quantum way. They are a collection of quantum gates that work together to produce quantum information and compute results from input information. They have, however, some limitations compared to a classical computational circuit, namely that they cannot use any feedback information (occurrences where newer bits influence previous bits), there can not be any non-reversible operations, and there can not be any copying of qubits. These circuits can be designed to perform specific tasks and are used to perform quantum algorithms [16].

An extension of quantum circuits are the variational quantum circuits, quantum-classical circuit hybrids where some of the circuit's parameters are classically tunable. The circuit can therefore be trained on a classical computer by optimizing it iteratively to minimize a predetermined loss function. For this optimization process, a gradient-based optimization method is commonly used. This allows for variational quantum circuits to have smaller circuit depths compared to normal quantum circuits and are therefore less impacted by noise. They also become more flexible and can be used in all quantum computing tasks [17].

2.5 Quantum-Classical SVM Hybrid

A quantum-classical SVM hybrid is a classical SVM that utilizes some aspects of quantum computing [18]. The approach used in this study was the usage of quantum kernels instead of classical kernels. This hybrid variant of a SVM still computes the separating hyperplane in the same fashion as a classical SVM, the only difference being that the kernel value is computed on a quantum computer, or simulated in a quantum space on a classical computer.

2.5.1 Quantum Embedding Kernels

Quantum Embedding Kernels are very similar to classical kernels as they both compute the inner product between datapoints, with the key difference being that quantum kernels calculates this inner product in a Hilbert space. The datapoints from the input space are first mapped to their corresponding quantum state using a quantum feature map ψ. This map ψ can be defined using a quantum circuit $U_\psi(\vec{x})$ as follows:

$$|\psi(\vec{x})\rangle = U_\psi(\vec{x})\,|0\rangle^n \tag{6}$$

where n is the number of qubits needed to represent the datapoint in binary form [19]. The kernel value for a particular datapoint pair is then computed from the overlaps of the pair's quantum states, which is calculated according to the following equation:

$$k(\vec{x}_i, \vec{x}_j) = |\,\langle\psi(\vec{x}_i)|\psi(\vec{x}_j)\rangle\,|^2 = |\,\langle 0^n|\,U_\psi^\dagger(\vec{x}_i)U_\psi(\vec{x}_j)\,|0^n\rangle\,|^2 \tag{7}$$

However, due to possible noise, the embedded quantum state is not always pure. Therefore, the Hilbert-Schmidt inner product is often used instead, but Eq. (7) suffices as the study focuses on a noiseless environment, see Section 1.5 for more information.

2.5.2 Variational Quantum Kernels

Variational Quantum Kernels can be seen as extensions of quantum embedding kernels that employ a variational quantum circuit as its quantum feature map, meaning that some of the feature map's parameters can be optimized according to a loss function [20]. This optimization allows for the creation of a custom made kernel specifically suited for the desired classification task. A common loss function is the kernel alignment technique that maximises the kernel alignment - the similarity between the current kernel k and ideal kernel k_{ideal}. The kernel alignment is maximised as a high value of the kernel alignment indicates the existence of a data separation with a low generalization error [21]. In classification tasks, the ideal kernel matrix can be defined as $K_{ideal} = \mathbf{y}\mathbf{y}^T$, where $\mathbf{y} = (\vec{y}_1, ..., \vec{y}_l)$ and l is the amount of training datapoints. The kernel alignment can then be computed according to the following equation:

$$A(K, K_{ideal}) = \frac{\langle K, K_{ideal}\rangle_F}{\sqrt{\langle K, K\rangle_F \langle K_{ideal}, K_{ideal}\rangle_F}} = \frac{\mathbf{y}^T K \mathbf{y}}{l||K||_F} \tag{8}$$

Here, $||K||_F$ is the Frobenius norm and $\langle K_{ideal}, K_{ideal}\rangle_F$ is the Frobenius inner product between the two matrices, computed as [21]:

$$\langle K_1, K_2\rangle_F = \sum_{i=1}^{l}\sum_{j=1}^{l} K_1(\overrightarrow{x}_i, \overrightarrow{x}_j)K_2(\overrightarrow{x}_i, \overrightarrow{x}_j) \tag{9}$$

2.6 Related Work

There have been several scientific papers written about the implementation of quantum computing in machine learning. Here are some research papers that were key to this paper.

2.6.1 An introduction to quantum machine learning

In this review article by Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione, published in 2015, prior research around quantum machine learning and its parts are presented. It discusses the different ways quantum machine learning can be implemented, notably either through finding dedicated quantum algorithms or through using the inherent probabilistic qualities of quantum theory to get better results for stochastic problems. It discusses the fact that both machine learning and quantum knowledge are both well explored but the union of these two fields still has not come far. The paper also provides information which indicates that quantum kernels and SVMs are faster at evaluating inner products in machine learning training vector spaces. It concludes with optimism around the subject and stating that further developments in the field may lead to great technological leaps [2].

2.6.2 Quantum Support Vector Machine for Big Data Classification

This research article written by Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd, was published in 2014 and theorises about Support Vector Machine and their potential benefits. Through mathematical reasoning, the authors show that an SVM can be implemented through quantum mechanics and that it could be beneficial future quantum ”big data” processors. It presents how a quantum implementation could lead to a speed-up with varying strength depending on the input data. The data privacy brought forward by this solution to the SVM problem is also presented, though this is not much of interest to our paper [7].

2.6.3 Kernel methods in Quantum Machine Learning

Riccardo Mengoni and Alessandra Di Pierro’s review article, published in 2019, discusses many different usages of quantum computing for kernel methods. The focus is on Classical-Quantum algorithms which are classical algorithms which use quantum computation to attain a speedup. The article states advantages and drawbacks of some different ways of handling this issue, with the authors splitting up the current day research into two different categories. The ways this article summarizes quantum kernels and their use in SVMs is of particular use to our thesis since it is very related. The article also has a very well structured table of research for other research papers in this field, some of which are of great use to a thesis like this one [22].

2.6.4 Supervised learning with quantum-enhanced feature spaces

In this research letter by Vojtěch Havlíček (et al.), published in 2019, a quantum-enhanced feature space is used to experimentally implement a pair of quantum algorithms. These quantum algorithms are used to construct classifiers. The two different methods of implementing the classifier are used to explore potential application of quantum computers within machine learning. The authors experimentally implement these methods and test their function to a successful result. The conclusion of the experiments are that for a quantum advantage to formally exist a feature map that is difficult to estimate classically would be an important factor. They also bring forth the issue with noise and how it can have a severe effect on the success of a hybrid classical-quantum algorithm [19].

2.6.5 Quantum machine learning in feature Hilbert spaces

The research article *Quantum machine learning in feature Hilbert spaces* authored by Maria Schuld and Nathan Killoran, published in 2019, introduces the similarities between the base building blocks of quantum computing and machine learning, and a way of harnessing this to produce efficient quantum machine learning through quantum kernels is theorized. Two forms of quantum kernels for machine learning classifiers are proposed and illustrated through math and simulations. Quantum states inside feature spaces are used to evaluate these kernel functions, and the small scale simulations provided ”interesting results”[23]. It also proceeds to discuss the fact that this kind of research is still in its infancy and that there is much more to research about pertaining to this subject [23].

3 Method

The method section of this study provides a detailed explanation of the methodologies employed to answer the research questions. This includes the reasoning behind why certain variables and algorithms were used, as well as explanations for fundamental theory behind each step.

3.1 Methodology

To conduct this study, a project was developed in the Python 3.9 programming language, using mainly the Scikit-learn ([4]) and Qiskit ([5]) software libraries. The project implements two SVMs using a classical and quantum kernel, respectively, as described in Section 3.4. The SVM implementations were tested on classical hardware, with the quantum kernel being simulated using the simulation method detailed in Section 3.2. To ensure the performance of both SVMs were comparable, both classifiers were trained and tested on the same datasets, which are described in more detail in Section 3.3. An overview of the process is illustrated in the flow chart in Figure 3.1 and the Python code used can be seen in the `quantum-classical-evaluation` GitHub repository [24]. The specifics are explained in detail in the following subsections.

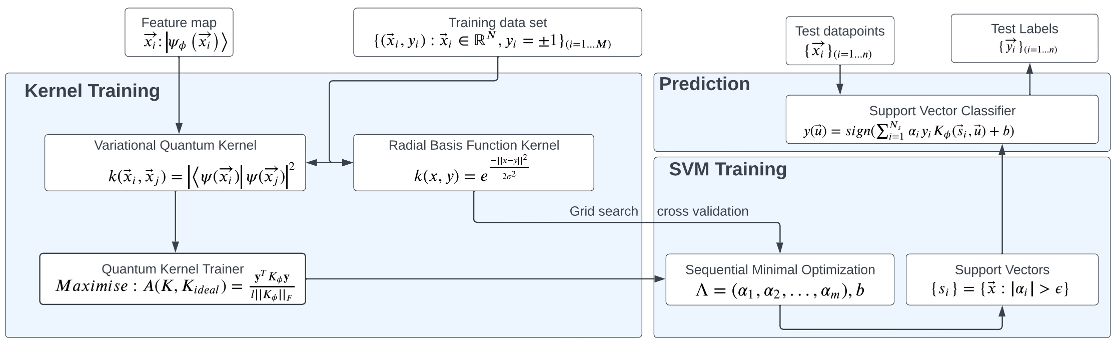

Feature map
$\vec{x_i} : |\psi_\phi(\vec{x_i})\rangle$
Training data set
$\{(\vec{x}_i, y_i) : \vec{x}_i \in \mathbb{R}^N, y_i = \pm 1\}_{(i=1\dots M)}$
Test datapoints
$\{\vec{x_i}\}_{(i=1\dots n)}$
Test Labels
$\{\vec{y_i}\}_{(i=1\dots n)}$
Kernel Training
Prediction
Support Vector Classifier
$y(\vec{u}) = sign(\sum_{i=1}^{N_s} \alpha_i y_i K_\phi(\vec{s}_i, \vec{u}) + b)$
Variational Quantum Kernel
$k(\vec{x}_i, \vec{x}_j) = |\langle \psi(\vec{x_i}) | \psi(\vec{x_j}) \rangle|^2$
Radial Basis Function Kernel
$k(x, y) = e^{\frac{-||x-y||^2}{2\sigma^2}}$
SVM Training
Grid search
cross validation
Quantum Kernel Trainer
$Maximise : A(K, K_{ideal}) = \frac{\mathbf{y}^T K_\phi \mathbf{y}}{l||K_\phi||_F}$
Sequential Minimal Optimization
$\Lambda = (\alpha_1, \alpha_2, \dots, \alpha_m), b$
Support Vectors
$\{s_i\} = \{\vec{x} : |\alpha_i| > \epsilon\}$

Figure 3.1: A flowchart of the used pipeline split into three parts: kernel training, SVM training and prediction. In the kernel training section, a VQK and RBF kernel is constructed and their trainable parameters are optimised for the training dataset. The optimised kernels are then used in their respective SVMs to generate the support vectors. The support vector classifier then uses these support vectors to predict the class of the test datapoints and assigns them their predicted labels.

To evaluate the classifiers, functions from Scikit-learn will be employed to compute certain parameters from the test data predictions, such as accuracy and precision, and Yellowbrick ([25]) will be used to plot these parameters in a clear way. The chosen parameters to base the evaluation on are collected in Table 3.3, with further explanations about the chosen parameters in Section 3.5, and are inspired by the work of Peter Flach [3].

3.2 Simulation of Quantum computer

All quantum computing in the program was run entirely using the Qiskit library's quantum simulators, not quantum hardware. The reasoning being that quantum hardware would be both difficult to access and lead to large amounts of noise that would make the comparison more difficult. To produce simulated quantum results on classical hardware, the statevectors were computed algebraically through the use of 2^n size complex vector matrices. By calculating the Schrödinger wave function of the simulated statevectors in the qubits, and transforming the values as the algorithms are performed, the values of the statevectors are attained. Though this can quickly gets very memory expensive, the low amount of qubits used in the project meant at largest a 3 by 3 matrix was used and the computation time was therefore reasonable. Due to this, statevector quantum simulation became an effective way to perform quantum computations on a classical computer in a noiseless setting [26].

3.3 Generating the data

The data used for this study was artificially generated by the `ad_hoc_data` function from the Qiskit software library, inspired by the work in Havlíček et. al. [19]. This function generates synthetic datasets using simulated quantum circuits while allowing for a great deal of customization in regards to the dataset properties, which were used to create datasets with different sizes and training to test datapoint ratios, these ratios can be seen in Table 3.1. These datasets had the additional benefit of inherently being linearly separable by the VQK, and not requiring any advanced reprocessing so they could be used immediately upon creation by both types of kernels. It was decided to generate this data with 3 features since more would have lead to an exponential increases in memory use and simulation time from the quantum simulations. The generated data can be seen in A.

Table 3.1: Table showing the amount of datapoints used for a run, as well as its training and testing split. The table also shows the number of repetitions a dataset with this setup was run and evaluated.

Data Amount	Training	Testing	Repetitions
100	80	20	5
90	80	10	1
80	60	20	2
80	40	40	1
50	40	10	2

3.4 Implementing The Classifiers

The SVM classifier used in this study was imported from the Scikit-learn library, which utilizes Sequential Minimal Optimization to solve the dual optimization problem and was set to use a soft-margin. This SVM implementation was chosen due to its compatibility with numerous Scikit-learn functions and other libraries, as well as its ability to handle custom-made kernels. The implemented kernels were also constructed using libraries. The VQK was built and trained using Qiskit functions and objects while the RBF kernel was a pre-built kernel in Scikit-learn with customizable parameters. The entire implementation process is illustrated in pseudo-code in Alg.(1) with the training process detailed in Section 3.4.1.

Algorithm 1 Overview of the method employed in this study to evaluate and compare two SVMs with a VQK and RBF kernel, respectively.

```
for $dataset \in Datasets$ do
    $X\_train, y\_train, X\_test, y\_test \leftarrow train\_test\_split(dataset)$

    $\#fm = feature\,map, tp = trainable\,parameters\#$
    $fm, tp \leftarrow create\_trainable\_fm(len(Xtrain))$
    $quantum\_kernel \leftarrow QuantumKernel(fm, tp)$
    $cb\_qkt = QKTCallback()$
    $spsa\_opt = SPSA(cb\_qkt.callback)$
    $qkt = QuantumKernelTrainer(quantum\_kernel, spsa\_opt)$
    $qka\_results = qkt.fit(X\_train, y\_train)$
    $optimized\_quantum\_kernel \leftarrow qka\_results.quantum\_kernel$
    $quantum\_SVM \leftarrow SVC(optimized\_quantum\_kernel)$

    $grid \leftarrow GridSearchCV(SVC(RBF))$
    $grid.fit(X\_train, y\_train)$
    $rbf\_SVM \leftarrow grid.best\_estimator\_$

    $quantum\_res \leftarrow eval(quantumSVM, X\_test, y\_test)$
    $rbf\_res \leftarrow eval(rbf\_SVM, X\_test, y\_test)$
    $save(quantum\_res, rbf\_res)$
end for
```

3.4.1 Training the kernels

The training process revolved around hyperparameter tuning for the kernels employed in the SVMs for the given dataset. For the classical SVM, cross-validation was performed to find close to optimal values for the C and σ variables in the RBF kernel. As seen in Alg.1, the cross-validation was made using Scikit's `GridSearchCV` function with Scikit's cross-validation strategy `StratifiedKFold`, which splits the training data into K equally large folds while preserving the class distribution. The `GridSearchCV` then takes K iterations and in each iteration, a new fold is used as the validation set and the rest are used to train the classifier, which will then be tested on the current iteration's validation set as seen in Fig 3.2. After all iterations have been completed, the optimal parameters can be computed from the results. This approach is useful in cases where the sample data is quite small, which is the case for this study, since the data usage is very efficient as the data folds can be used multiple times. The chosen value for K was 10 since it is a standard value for the amount of folds in cross-validation. The training on the RBF kernel was done to make for a fairer comparison to the variable quantum kernel that gets an optimized feature map.

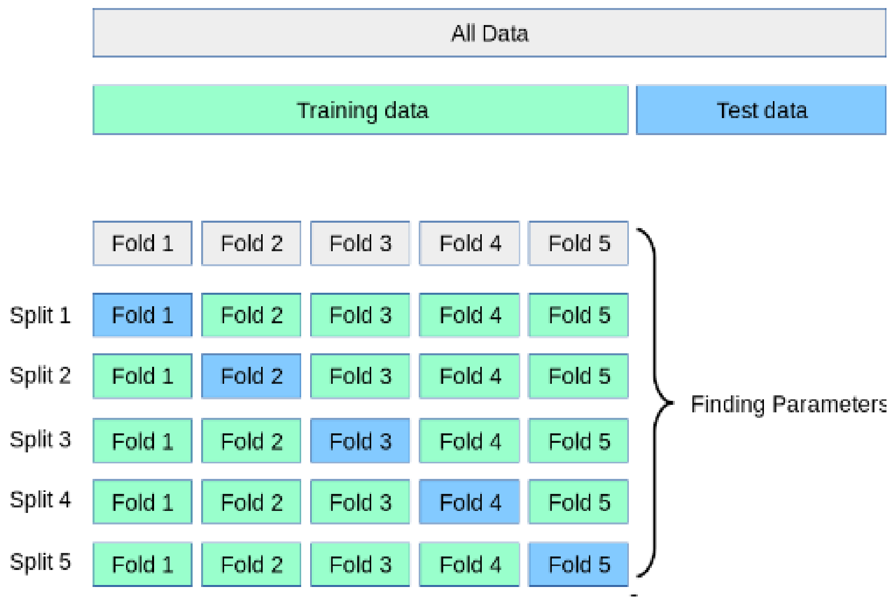
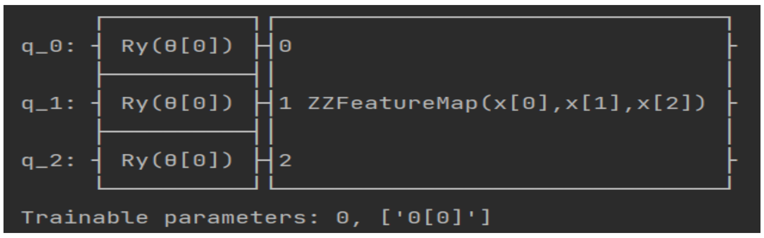


Figure 3.2: The figure demonstrates how Qiskit's `StratifiedKFold` cross-validation strategy works when $K = 5$. Image sourced from L Jian, et. al. ([27]).

For the VQK, a kernel with a trainable feature map was constructed and trained to find the optimal feature map for the given dataset. The feature map consists of a layer of RY gates followed up by a `ZZFeatureMap`, as seen in Figure 3.3, where the RY gates' rotation angle is a variable and can thus be optimized. The kernel used was Qiskit's `TrainableFidelityQuantumKernel`, which computes the overlap of two quantum states defined by the given encoding feature map, and it was trained using Qiskit's `QuantumKernelTrainer` function. The `QuantumKernelTrainer` was set to use the Quantum Kernel Alignment technique which iteratively trains the VQK, gradually converging to the kernel with the largest margin - the ideal kernel. This technique was performed by utilizing Qiskit's optimizer strategy `SPSA` to find the optimal rotation angle that gave the smallest hinge loss value, which is computed from the following equation:

$$L(y, f(x)) = max(0, 1 - y \cdot f(x)) \tag{10}$$

where y is the true label, f(x) is the predicted label and the max function is used to only penalize miscalculations.



```
     ┌──────────┐┌──────────────────────────────┐
q_0: ┤ Ry(θ[0]) ├┤0                             ├
     ├──────────┤│                              │
q_1: ┤ Ry(θ[0]) ├┤1 ZZFeatureMap(x[0],x[1],x[2]) ├
     ├──────────┤│                              │
q_2: ┤ Ry(θ[0]) ├┤2                             ├
     └──────────┘└──────────────────────────────┘
Trainable parameters: θ, ['θ[0]']
```

Figure 3.3: Figure demonstrating the structure of the trainable feature map.

3.5 Evaluating The Classifiers

To evaluate the performance of the classifiers, a number of evaluation metrics were chosen and the chosen metrics are described in Table 3.3.

The performance of the RBF SVM and Quantum-Classical SVM Hybrid were evaluated on the same testing dataset for each of the benchmark datasets. The Yellowbrick ([25]) software library, which builds upon scikit-learn, was used to calculate and display the results of the experiments. To compare the two kernels, the results from each SVM were displayed in a classification report, a receiver operating characteristic (ROC) curve, a confusion matrix, and a precision recall curve. These all provide information about specific performance in the classifiers. In our results mainly the data from the classification reports and the ROC curves were considered as they showed the most comparable data, though all of the plots and matrices can be found in the `quantum-classical-evaluation` GitHub repository [24].

3.5.1 Classification Report

The classification report, calculated and displayed by a `ClassificationReport` class from Yellowbrick, displayed several important metrics for each class in the classifier. It displayed a value and heatmap for four attributes:

Table 3.2: Attributes in the classification report.

Attribute	*Purpose*
Precision	**Used** to see what proportion of positive identifications that were correct. A model with no false positives has a precision of 1.00, a model with only false positives has a precision of 0.00.
Recall	**Used** to see what proportion of positive identifications that actually were made. A model with no false negatives has a recall of 1.00, a model with only false negatives has a recall of 0.00.
F1-score	**The** harmonic mean of precision and recall. Scores are between 0.00 and 1.00 and are based on the scores of precision and recall.
Support	**The** amount of samples the class contains.

3.5.2 Receiver Operating Characteristic Curve / Area Under Curve

The reciever operating characteristic curve, calculated and displayed by a `ROCAUC` class from Yellowbrick, displayed two very connected metrics about the classifier. It was used to measure the classifiers predictive quality by calculating the tradeoff between its specificity and sensitivity. The area under this curve was also calculated and is a measure of the relationship between true and false positives, called the AUC. Generally, the higher the value the better the model.

3.5.3 Confusion Matrix

The confusion matrix, calculated and displayed by a `ConfusionMatrix` class from Yellowbrick, displayed the true and false positives and negatives found by the classification algorithm. It was used to display numerically how many classifications the algorithm got right. The information from this matrix was used to calculate the accuracy of the model.

3.5.4 Precision Recall Curve

The precision-recall curve, calculated and displayed by a `PrecisionRecallCurve` class from Yellowbrick, displayed the tradeoff between precision and recall in a curve where the axis are the two metrics. It did this for different thresholds. A tall curve was desirable for both high precision and high recall.

3.5.5 Compiled table of evaluated metrics

Table 3.3: Evaluated parameters, inspired by the work of Peter Flach [3].

Attribute	*Purpose*
Accuracy	**The** ratio of correct answers over total answers.
Precision	**Used** to see what proportion of positive identifications that were correct. A model with no false positives has a precision of 1.00, a model with only false positives has a precision of 0.00.
Recall	**Used** to see what proportion of positive identifications that actually were made. A model with no false negatives has a recall of 1.00, a model with only false negatives has a recall of 0.00.
F1-score	**The** harmonic mean of precision and recall. Scores are between 0.00 and 1.00 with 1.00 being the best, and the scores are based on precision and recall.
AUC	**AUC** (Area under curve) is the area under a ROC (Reciever operating characteristic) curve. It is an aggregate measure of the performance of the classifier by computing the relationship between true and false positives.

4 Results

The following are the collected results from executions of our method. The results are split into one section concerning the VQK (Section 4.1) and one section concerning the RBF kernel (Section 4.2). Each section begins with a table of general data collected from a number of processes with different parameters, followed by a smaller subset of runs with the same parameters presented with further resulting data. These sections are followed by a section containing both results put together in comparison tables and figures. The amount of training and testing data is structured as n/m where n is for training and m is for testing. The splits can also be seen in Table 3.1.

4.1 Variational Quantum Kernel

Table 4.1: Results for separate runs in 3 dimensions on the VQK

Data	Precision	Recall	F1-Score	Accuracy	AUC
Averages	0.76	0.74	0.73	0.72	0.83
80/20	0.70	0.68	0.67	0.68	0.81
80/20	0.80	0.80	0.80	0.80	0.91
80/20	0.78	0.78	0.78	0.53	0.85
80/20	0.80	0.78	0.77	0.78	0.86
80/20	0.80	0.80	0.80	0.80	0.77
80/10	0.74	0.70	0.69	0.70	0.85
60/20	0.80	0.78	0.77	0.78	0.84
60/20	0.68	0.68	0.68	0.65	0.77
40/40	0.91	0.90	0.90	0.90	0.97
40/10	0.81	0.70	0.67	0.70	0.91
40/10	0.56	0.55	0.55	0.55	0.59

Table 4.2: Results for 80/20 runs in 3 dimensions on the VQK

80/20	Precision	Recall	F1-Score	Accuracy	AUC
Averages	0.77	0.77	0.76	0.72	0.84
Run 1	0.70	0.68	0.67	0.68	0.81
Run 2	0.80	0.80	0.80	0.80	0.91
Run 3	0.78	0.78	0.78	0.53	0.85
Run 4	0.80	0.78	0.77	0.78	0.86
Run 5	0.80	0.80	0.80	0.80	0.77

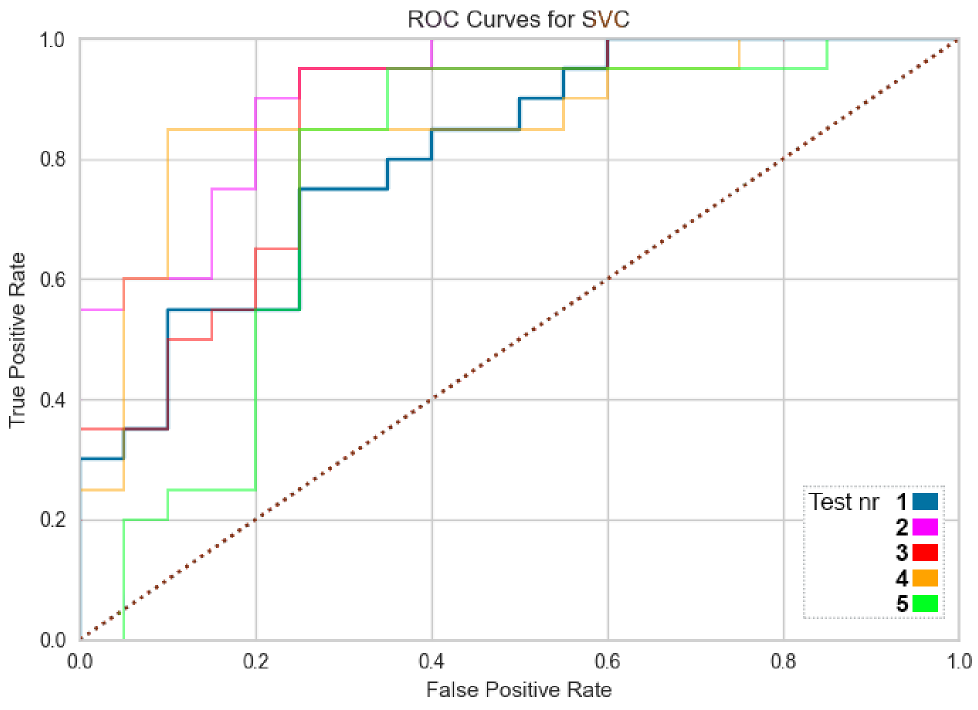


Figure 4.1: The ROC curve for the five runs of the VQK with 80/20 data split. The exact value for the AUC is in Table 4.2.

4.2 RBF Kernel

Table 4.3: Results for separate runs in 3 dimensions on the RBF kernel.

Data	Precision	Recall	F1-Score	Accuracy	AUC
Averages	0.54	0.53	0.49	0.53	0.54
80/20	0.75	0.68	0.65	0.68	0.77
80/20	0.60	0.60	0.60	0.60	0.56
80/20	0.53	0.53	0.53	0.53	0.53
80/20	0.42	0.43	0.42	0.42	0.53
80/20	0.50	0.50	0.46	0.50	0.45
80/10	0.76	0.55	0.44	0.50	0.56
60/20	0.50	0.50	0.46	0.50	0.59
60/20	0.55	0.55	0.55	0.55	0.53
40/40	0.59	0.56	0.53	0.56	0.6
40/10	0.45	0.45	0.44	0.45	0.37
40/10	0.25	0.50	0.34	0.50	0.42

Table 4.4: Results for 80/20 runs in 3 dimensions on the RBF kernel

80/20	Precision	Recall	F1-Score	Accuracy	AUC
Averages	0.56	0.55	0.53	0.55	0.57
Run 1	0.75	0.68	0.65	0.68	0.77
Run 2	0.60	0.60	0.60	0.60	0.56
Run 3	0.53	0.53	0.53	0.53	0.53
Run 4	0.42	0.43	0.42	0.42	0.53
Run 5	0.50	0.50	0.46	0.50	0.45

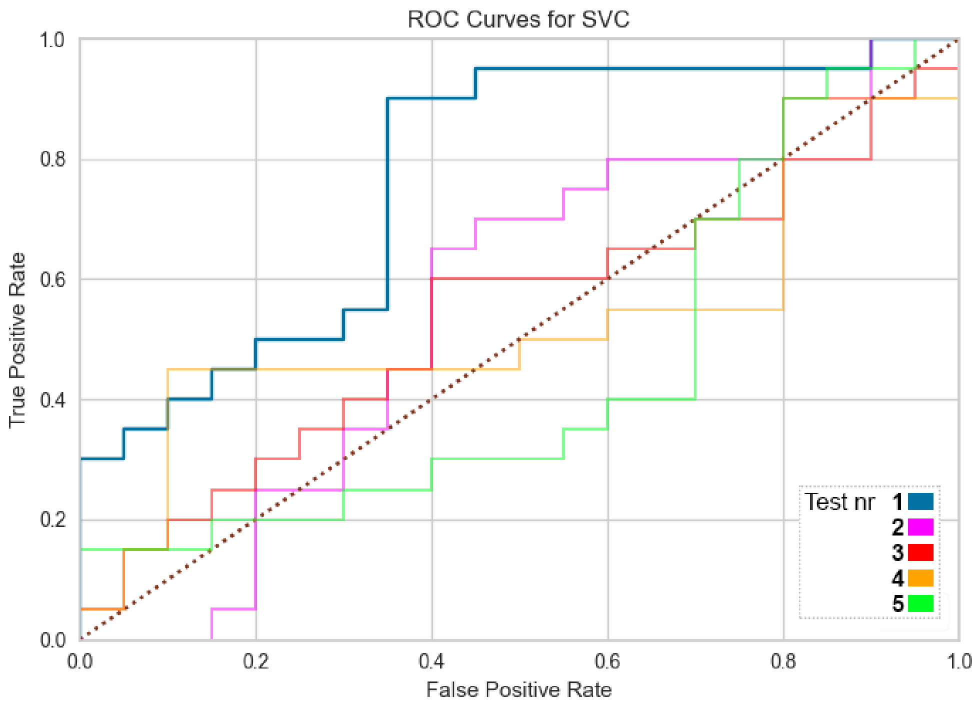


Figure 4.2: The ROC curve for the five runs of the RBF kernel with 80/20 data split. The exact value for the AUC is in Table 4.4.

4.3 Comparison

The following section contains values from both kernels when put directly head to head. This means that for every value, the resulting score is computed using the equation $quantum\_score - rbf\_score$. In Table 4.3, a positive value means the VQK has the edge whilst a negative value means the RBF kernel has the edge. The kernels have been assigned colours, with the VQK being red and the RBF kernel being blue, for visual clarity.

Table 4.5: Difference in results between both kernels for separate runs in 3 dimensions. The kernels have assigned colours, with the VQK being red and the RBF being blue, and the stronger the colour, the bigger the advantage.

Data	Precision	Recall	F1-Score	Accuracy	AUC
Averages	0.22	0.21	0.24	0.19	0.29
80/20	−0.05	0.00	0.02	0.00	0.04
80/20	0.20	0.20	0.20	0.20	0.35
80/20	0.25	0.25	0.25	0.00	0.32
80/20	0.38	0.35	0.36	0.36	0.33
80/20	0.30	0.30	0.35	0.30	0.32
80/10	−0.03	0.15	0.25	0.20	0.29
60/20	0.30	0.28	0.32	0.28	0.25
60/20	0.13	0.13	0.13	0.10	0.24
40/40	0.33	0.34	0.37	0.34	0.37
40/10	0.37	0.25	0.24	0.25	0.54
40/10	0.31	0.05	0.22	0.05	0.17

Table 4.6: Difference between the VQK and RBF kernel in 80/20 runs in 3 dimensions. The kernels have assigned colours, with the VQK being red and the RBF being blue, and the stronger the colour, the bigger the advantage.

80/20	Precision	Recall	F1-Score	Accuracy	AUC
Averages	0.21	0.22	0.23	0.17	0.27
Run 1	−0.05	0.00	0.02	0.00	0.04
Run 2	0.20	0.20	0.20	0.20	0.35
Run 3	0.25	0.25	0.25	0.00	0.32
Run 4	0.38	0.35	0.36	0.36	0.33
Run 5	0.30	0.30	0.35	0.30	0.32

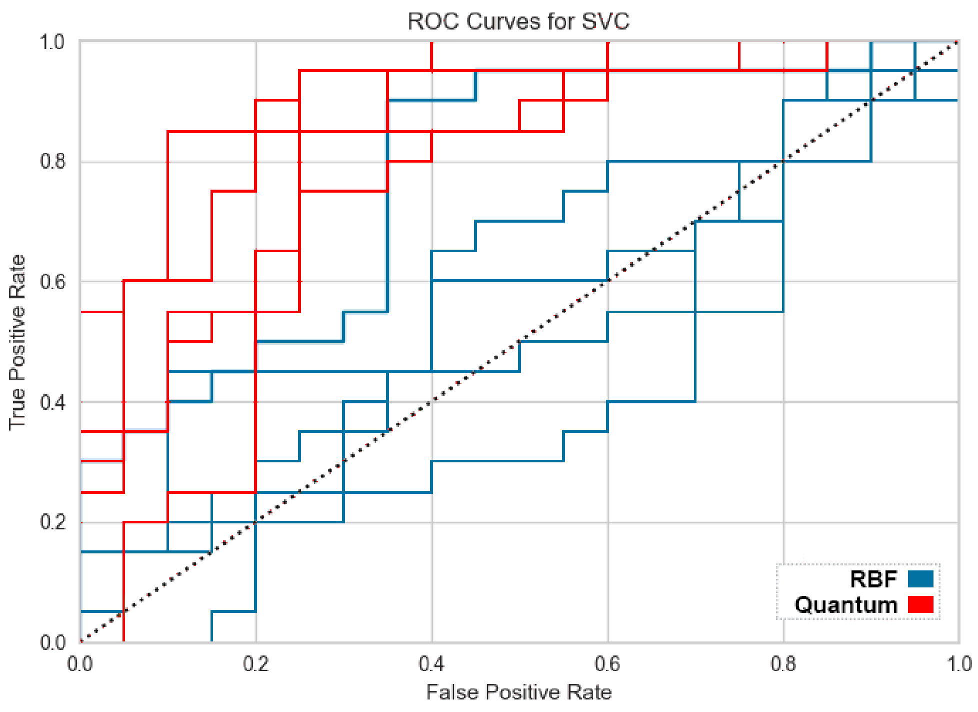


Figure 4.3: The ROC curve for the five runs of both with 80/20 data split. The exact value for the AUC is in Table 4.6.

5 Discussion

5.1 Summary of results

In general, the results in Table 4.5 present a clear picture of the quantum kernel (VQK) being superior to the classical (RBF) kernel in most performance metrics on the specific data used in this research paper. Only in a few cases, namely the precision of the first 80/20 run and the 80/10 run, does the performance of the classical kernel surpass that of the quantum kernel. Due to how other runs with the same parameters show a result more similar to the average, these few cases can be counted as outliers in our results.

In most cases, the improvement of the quantum kernel over the classical kernel are quite noticeable. On average, only the accuracy of the quantum kernel has a less than 0.20 unit advantage over the classical kernels. This lead is 0.19 units, however, so the average advantage is still considerably high. When looking at the comparison numbers for the five 80/20 runs in Table 4.6, we can see that the largest advantage between the averages of the two kernels is the quantum kernel's 0.84 units of AUC, which is 0.27 units higher than the classical kernel's 0.57 units in the same statistic. This lead can also be visualized in Figure 4.3, where the ROC curves of the quantum kernel runs generally being more optimal than the ROC curves of the classical kernel. As the position and slope of this curve directly affects the AUC value, this chart visualizes the reason for the large discrepancy between the AUC value between the quantum and classical kernels.

5.2 Interpretation of results

If the results seen in Table 4.5 would be taken at face value, it would seem obvious that the quantum kernel performs better than the classical kernel within the tested metrics. It is, however, not entirely possible to directly translate the results into general or real world use. What our research proves is that when using SVMs with these specific kernels optimised in this one way to classify this specific kind of quantum data, there is quite a large performance gain possible by using a simulated quantum kernel over a classical kernel. It is nonetheless important to consider other factors related to our research that were not possible to test due to the simulated nature of our methodology. For one, we did not consider the time needed to train the kernels as this would be slower on a simulated quantum computer compared to actual physical hardware, due to all the necessary calculations for the simulation. There is also the issue of noise that would appear if the model was run on a real quantum computer - something that we did not include in our results since we only researched the potential of the kernel in a simulated noiseless environment. These factors could affect the standing of the quantum kernel over the classical kernel. There is also the limitations and sources of errors we discuss in further detail in Section 5.4.

What the research proves, however, is that there are potential scenarios where a quantum kernel can match and exceed the general performance of a classical kernel. When using our SVMs to classify the generated data, the quantum kernel outperforms the classical kernel, indicating that there may be other scenarios too where this can be the case. It tells us that it is worth to continue research on this novel topic since if there is potential benefits within artificial datasets, there may be some within real data too.

When it comes to previous research, our results support earlier findings. The papers presented in Section 2.6 are generally optimistic about the potential performance increases that can be had from quantum kernels, and our findings support these claims. For example, our research corroborates with the information in [7] (which is summarized in Section 2.6.2) that the speedup could depend on the input data. It also aligns with [19], summarized in Section 2.6.4, in considering that quantum kernels can produce a quantum advantage, at least when not considering noise.

5.3 Implication of the research

It is difficult, if not impossible, to know the true implication of what quantum machine learning could mean for technology in the future. However, since we found a potential performance advantage in the use of quantum kernels over classical kernels, we could theorise about what such a performance increase could imply.

For now, the development of a truly superior quantum algorithm would not affect the world too much. The field of quantum computing is still in its late infancy and there is just not enough quantum computers with the kind of performance required to utilize this research at a larger scale. Things are changing, however, and with time comes more plentiful and more powerful quantum computers. The development of a superior algorithm for machine learning classification could definitely spur interest within the industry to put even more investments into developments in the quantum computing field. This could have numerous benefits, as it could allow for even more exciting and usable applications to be discovered. Since the conducted research also only utilizes three qubits, there is a possibility that this could be used even on the most basic variants of any newly developed quantum hardware.

If quantum computing is developed enough for general users to be able to utilize the performance increase it offers, the implication of a quantum advantage in machine learning classifiers would be immense. A large amount of machine learning performed in this day and age is based upon classifying and labeling data, something that could achieve better performance with the help of quantum computers. This could mean better performing search engines, personal recommendations, voice recognition, self driving vehicles and more. It would not only provide better predictions but possibly even make the whole endeavour more environmentally sustainable through lower power use. This is assuming that future quantum computers are on par with modern classical computers when it comes to power efficiency.

5.4 Limitations and sources of errors

The limitations of the research and results gathered are very connected to and often the same as the sources of errors. The most major limitation presented throughout the research was the type of data used for the testing. The synthetic datasets generated by the `ad_hoc_data` function were easy to handle and work with, and performed well in our scenarios. The fact that the data was artificially generated from a single source does, however, limit the inferences we can pull from the research results. It gives a potential edge to the quantum kernel due to the quantum nature of its generation and the fact that the generation method guarantees the data to be linearly separable by a quantum kernel. Thus, the implications from our results may not demonstrate the whole truth.

Additionally, the amount of time available limited our ability to generate more substantial amounts of results. Every execution of the program took a considerable amount of time for the quantum-classical SVM hybrid, which increased exponentially with the size of the datasets. Therefore, it was not possible for us to test and compare our models on large datasets and thus we restricted ourselves to datasets with a maximum size of 100 datapoints. This means that the results in this paper only give a rough pointer to the real world performance, since getting the proper value would require substantial amounts of runs with datasets of varying sizes.

Furthermore, our study focused on comparing quantum and classical kernels in SVMs and the kernel usage was restricted to only two kinds: the classical RBF kernel and the VQK. It is therefore important to note that while our findings did indicate a potential advantage of quantum kernels over classical kernels, the same might not be true for other classification models or even other SVMs using different kernels.

Finally, potential errors and limitations spring from the optimisation of the kernels. The time and effort required to optimize the kernels to a point of diminishing returns would have been too immense and out of the possible scope for our research, but the results may have suffered due to this. It is possible that the considerable performance difference between the VQK and the RBF kernel is due to the different ways these kernels were optimised. This may have lead to us optimising the kernels in an uneven way, with the quantum kernel receiving more attention than the classical one, leading to the performance of the quantum kernel outweighing that of the classical kernel.

5.5 Future research

Due to the methodology used in this research and the results we gathered, there is a lot of potential for future research that would build upon the researched subject. Listing all potential facets of research would be impossible, so here is a curated list of suggestions that the authors of this paper have thought about:

- Attempt the comparison with different kinds of input data, both real data and other forms of computationally generated data, to reduce the risk of the data being biased to one of the kernels.
- Repeat the comparison with kernels that have been fully optimised and data that has been preprocessed to get a fairer comparison.
- Extend the research to test with more than three features and two classes and see how that affects the performance.
- Compare different kinds of quantum kernels with different kinds of classical kernels to get a more general idea of performance differences.
- Evaluate quantum and classical kernels in other classification models that utilizes kernels to investigate the generalizability of the study's results.
- Perform the research on actual quantum hardware to compare performance when considering time and noise to figure out if the performance gain is utilizable.

6 Conclusions

In recent years, the field of quantum computing has seen a lot of progress. Through our research of quantum machine learning kernels, we have gotten a taste of that progress, and in a way become a part of it. Through the usage of specialized software libraries and qubit simulation, a classical and a quantum kernel have been compared. Although the research had quite a lot of limitations, we have still gotten an idea of how the performance of a classical kernel compares to a quantum kernel within binary classification tasks. We have seen how, at least within specific parameters, quantum kernels can have a significant lead over classical kernels when judging from some common classification metrics. We see that with the setup used in this paper, the quantum kernel consistently has the same or higher performance than the classical kernel. We also discussed how these results could be improved upon to give a more general analysis on the performance difference between the kernels, for example using real datasets instead of generated ones and by using physical quantum hardware.

Through extended discussion, we also theorised about the implications of the results regarding the potential use of quantum computing in machine learning. We have realized that a proven effective use of quantum computers within machine learning would both increase the performance of available machine learning algorithms, while also helping to spur on the increasingly active field of quantum computing. This could assist in the development of machine learning applications, such as search engines and self driving cars, but further research is needed to figure out how viable it may be. This is why we have recommended potential future topics to research, such as examining whether the way we generated the data strongly affected the performance, and inspecting other affecting factors like time and noise when it comes to doing the calculations on actual quantum hardware.

6.1 Final Words

While the performed work did not produce any groundbreaking results, the experiment is still interesting and further proves that quantum algorithms for machine learning may be viable in the future. Our results are in line with previous research which supports our findings, but further research is required to conclusively prove the quantum advantage, if it exists.

References

[1] Shor, P.W. “Algorithms for quantum computation: discrete logarithms and factoring”. In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700.

[2] Schuld, Maria, Sinayskiy, Ilya, and Petruccione, Francesco. “An introduction to quantum machine learning”. In: *Contemporary Physics* 56.2 (2015), pp. 172–185. DOI: 10.1080/00107514.2014.964942. eprint: https://doi.org/10.1080/00107514.2014.964942. URL: https://doi.org/10.1080/00107514.2014.964942.

[3] Flach, Peter. *Machine learning*. en. Cambridge, England: Cambridge University Press, Sept. 2012. ISBN: 978-1107422223.

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

[5] Qiskit contributors. *Qiskit: An Open-source Framework for Quantum Computing*. 2023. DOI: 10.5281/zenodo.2573505.

[6] James, Gareth et al. *An Introduction to Statistical Learning: with Applications in R, Second Edition*. Springer, Sept. 2021. URL: https://hastie.su.domains/ISLR2/ISLRv2_website.pdf.

[7] Rebentrost, Patrick, Mohseni, Masoud, and Lloyd, Seth. “Quantum Support Vector Machine for Big Data Classification”. In: *Physical Review Letters* 113.13 (Sept. 2014). DOI: 10.1103/physrevlett.113.130503. URL: https://doi.org/10.1103%2Fphysrevlett.113.130503.

[8] Moguerza, Javier M and Muñoz, Alberto. “Support vector machines with applications”. In: (2006).

[9] Cortes, Corinna and Vapnik, Vladimir. “Support-vector networks”. In: *Machine learning* 20 (1995), pp. 273–297.

[10] Burges, Christopher JC. “A tutorial on support vector machines for pattern recognition”. In: *Data mining and knowledge discovery* 2.2 (1998), pp. 121–167.

[11] Hearst, Marti A et al. “Support vector machines”. In: *IEEE Intelligent Systems and their applications*. IEEE, 1998, pp. 18–28.

[12] Ma, Hui, Ekanayake, Chandima, and Saha, Tapan K. “Power transformer fault diagnosis under measurement originated uncertainties”. In: *IEEE Transactions on Dielectrics and Electrical Insulation* 19.6 (2012), pp. 1982–1990.

[13] James, Gareth et al. *An Introduction to Statistical Learning: With Applications in R*. Springer, 2013.

[14] Scholkopf, Bernhard et al. “Comparing support vector machines with Gaussian kernels to radial basis function classifiers”. In: *IEEE transactions on Signal Processing* 45.11 (1997), pp. 2758–2765.

[15] Yanofsky, Noson S. and Mannucci, Mirco A. *Quantum Computing for Computer Scientists*. Cambridge University Press, 2008. DOI: `10.1017/CBO9780511813887`.

[16] Nielsen, Michael A. and Chuang, Isaac L. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010. DOI: `10.1017/CBO9780511976667`.

[17] Cerezo, Marco et al. “Variational quantum algorithms”. In: *Nature Reviews Physics* 3.9 (2021), pp. 625–644.

[18] Hubregtsen, Thomas et al. “Training quantum embedding kernels on near-term quantum computers”. In: *Physical Review A* 106.4 (Oct. 2022). DOI: `10.1103/physreva.106.042431`. URL: `https://doi.org/10.1103%2Fphysreva.106.042431`.

[19] Havlíček, Vojtěch et al. “Supervised learning with quantum-enhanced feature spaces”. In: *Nature* 567.7747 (Mar. 2019), pp. 209–212. ISSN: 1476-4687. DOI: `10.1038/s41586-019-0980-2`. URL: `https://doi.org/10.1038/s41586-019-0980-2`.

[20] Glick, Jennifer R et al. “Covariant quantum kernels for data with group structure”. In: *arXiv preprint arXiv:2105.03406* (2021).

[21] Wang, Tinghua, Zhao, Dongyan, and Tian, Shengfeng. “An overview of kernel alignment and its applications”. In: *Artificial Intelligence Review* 43 (2015), pp. 179–192.

[22] Mengoni, Riccardo and Di Pierro, Alessandra. “Kernel methods in Quantum Machine Learning”. In: *Quantum Machine Intelligence* 1.3 (Dec. 2019), pp. 65–71. ISSN: 2524-4914. DOI: `10.1007/s42484-019-00007-4`. URL: `https://doi.org/10.1007/s42484-019-00007-4`.

[23] Schuld, Maria and Killoran, Nathan. “Quantum Machine Learning in Feature Hilbert Spaces”. In: *Physical Review Letters* 122.4 (Feb. 2019). DOI: `10.1103/physrevlett.122.040504`. URL: `https://doi.org/10.1103%2Fphysrevlett.122.040504`.

[24] Jacob Westergren, Teo Nordström. *quantum-classical-evaluation*. `https://gits-15.sys.kth.se/teon/quantum-classical-evaluation`. 2023.

[25] Bengfort, Benjamin et al. *Yellowbrick*. Version 0.9.1. Nov. 14, 2018. DOI: `10.5281/zenodo.1206264`. URL: `http://www.scikit-yb.org/en/latest/`.

[26] Raedt, Koen De et al. “Massively parallel quantum computer simulator”. In: *Comput. Phys. Commun.* 176.2 (2007), pp. 121–136. DOI: `10.1016/j.cpc.2006.08.007`. URL: `https://doi.org/10.1016/j.cpc.2006.08.007`.

[27] Jian, L et al. “Rapid Analysis of Cylindrical Bypass Flow Field Based on Deep Learning Model”. In: *IOP Conference Series: Earth and Environmental Science* 1037 (June 2022), p. 012013. DOI: `10.1088/1755-1315/1037/1/012013`.

Appendices

A Dataset visualisations

Visualizations of the datasets used for the tests.

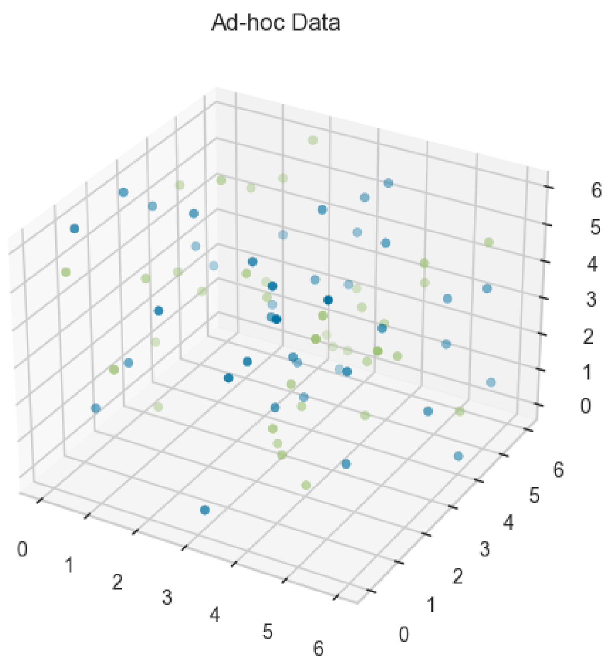


Figure A.1: Dataset used in the first run with a 40/10 split.

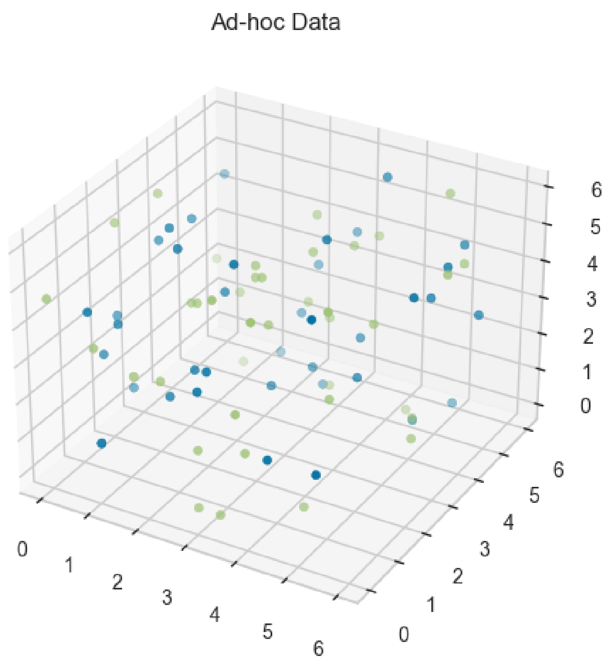


Figure A.2: Dataset used in the second run with a 40/10 split.

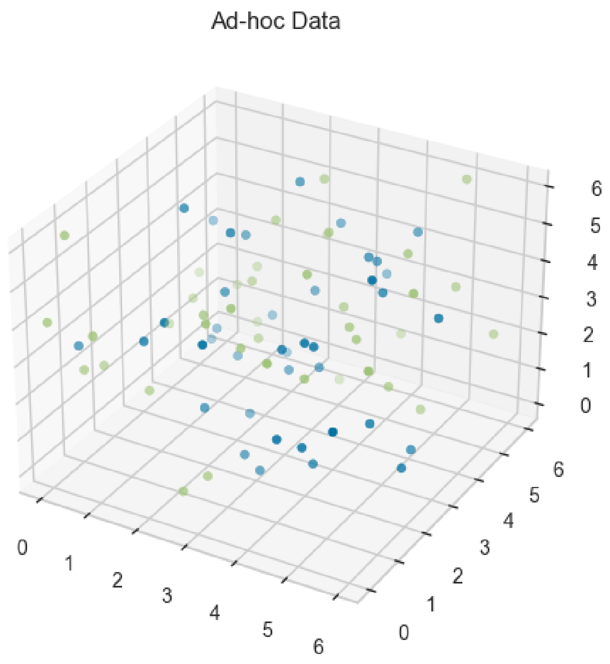


Figure A.3: Dataset used in the run with a 40/40 split.

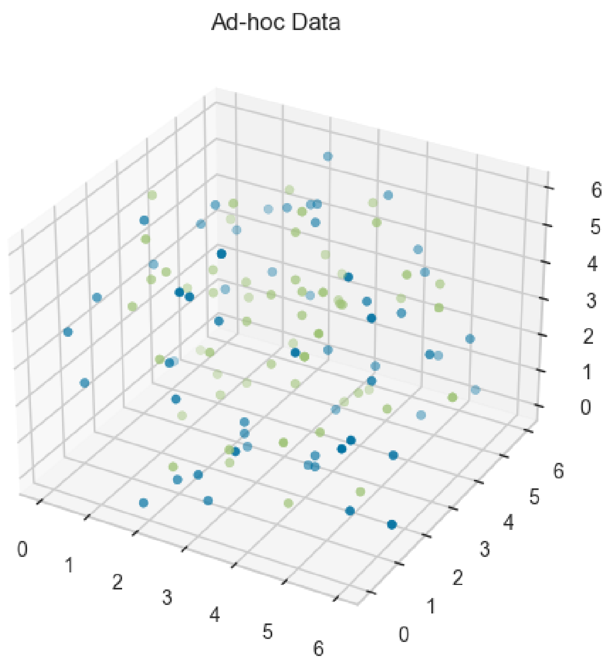


Figure A.4: Dataset used in the first run with a 60/20 split.

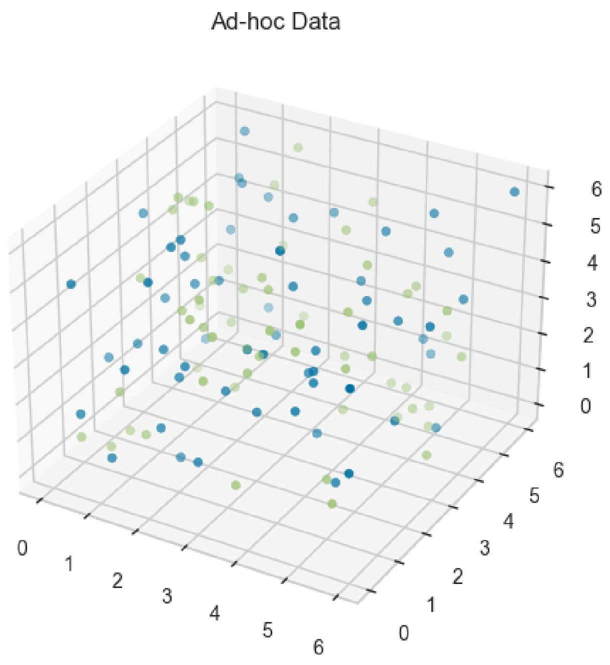


Figure A.5: Dataset used in the second run with a 60/20 split.

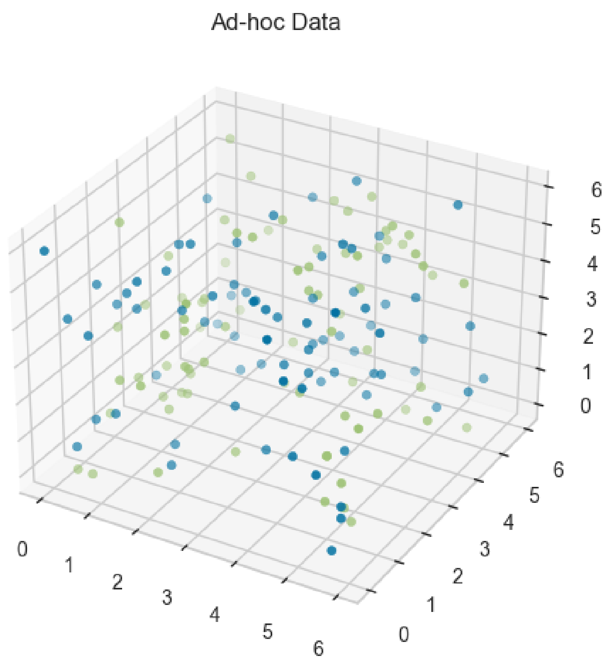


Figure A.6: Dataset used in the run with a 80/10 split.

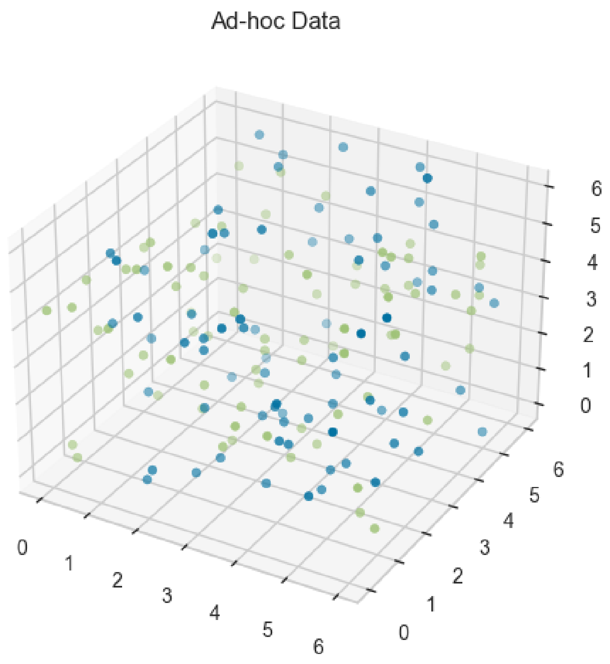


Figure A.7: Dataset used in the first run with a 80/20 split.

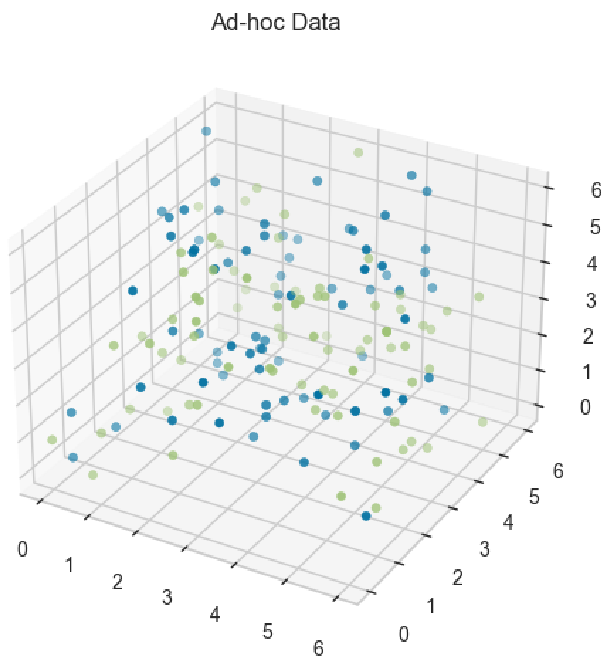


Figure A.8: Dataset used in the second run with a 80/20 split.

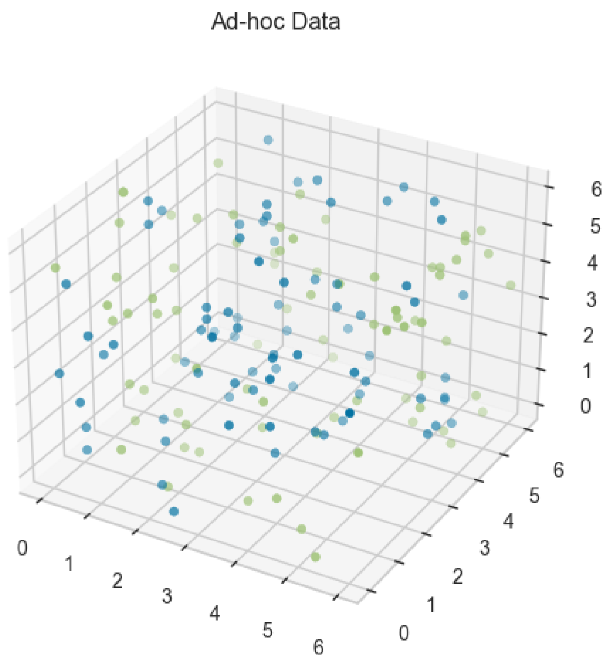


Figure A.9: Dataset used in the third run with a 80/20 split.

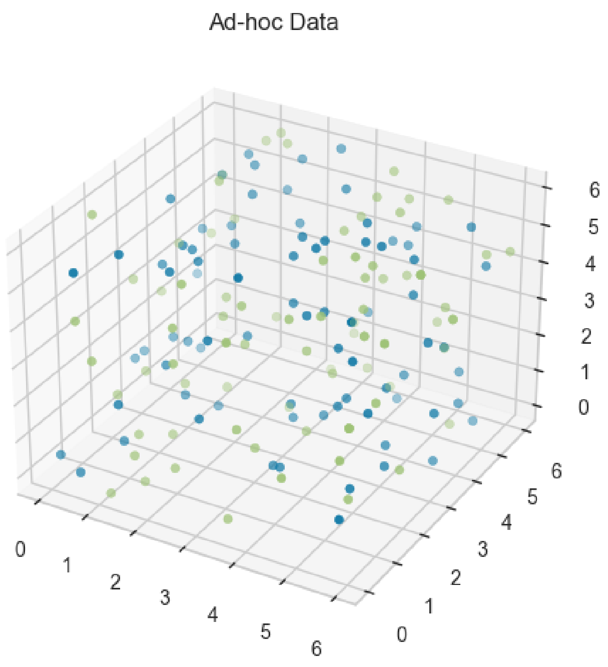


Figure A.10: Dataset used in the fourth run with a 80/20 split.

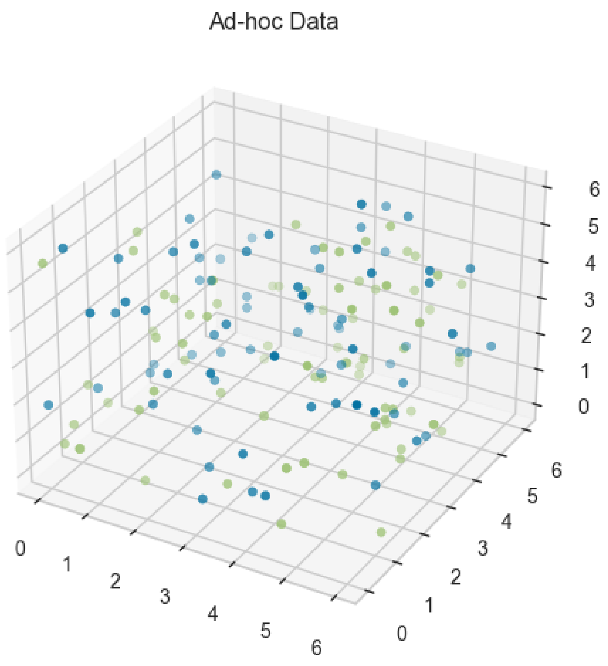


Figure A.11: Dataset used in the fifth run with a 80/20 split.

TRITA-EECS-EX-2023:332