MACHINE LEARNING USING BITS OR QUBITS: UNDERSTANDING THE ADVANTAGES OF QUANTUM MACHINE LEARNING

Judah Felix
School of Computing
Newcastle University, England, UK
J.F.Rajarathnam2@newcastle.ac.uk

ABSTRACT

As we stride into the quantum age, the rapid advancements in quantum computing have opened new frontiers for computational capabilities. Quantum computers harness the principles of quantum mechanics to perform computations at an unprecedented scale, promising exponential speedups over classical counterparts for certain tasks. Concurrently, machine learning has emerged as a transformative technology across various domains, driving innovations in data analytics, pattern recognition, and decision-making systems. With the advent of quantum computing, the prospect of leveraging quantum mechanics to enhance machine learning algorithms has become a tantalising possibility. Exploring machine learning on quantum computers holds the potential to unlock novel solutions to complex optimisation problems, enable more efficient data processing, and facilitate breakthroughs in artificial intelligence. In this research, we delve into the performance comparison between quantum machine learning and classical machine learning methodologies. We investigate the advantages and limitations of quantum ML algorithms, assess their scalability and robustness, and evaluate their efficacy in handling real-world datasets. Moreover, we highlight the significance of integrating classical and quantum hybrid models to harness the complementary strengths of both paradigms, paving the way for hybrid quantum-classical machine learning frameworks. Through empirical analysis and theoretical insights, this study elucidates the landscape of quantum machine learning, delineates its potential impact on various industries, and underscores the imperative of fostering interdisciplinary collaborations to propel the frontier of quantum-enhanced machine learning research.

***K*eywords** Quantum computing · Machine learning · Quantum machine learning · Performance comparison · Advantages · Hybrid models · Quantum age · Computational capabilities · Quantum mechanics · Optimisation problems · Scalability · Robustness · Artificial intelligence · Interdisciplinary collaborations · Quantum-enhanced machine learning

1 Introduction

As humanity advances into the quantum era and goes from using the word "quantum" for in-explainable science fiction in Marvel movies to the installation of the world's first room temperature quantum computer [39], we mark a transformative milestone in computational history. Quantum computers are processing machines that essentially function on the laws of quantum physics, which may be extremely beneficial for specific tasks, where they can beat the performance of the most powerful supercomputers installed today. A quantum computer could be considered to be the co-processor of a traditional computer. As shown in Figure 1.1.1, a classical computer can precisely control a quantum computer's operations by triggering qubit operations performed by the quantum gates at precise rates [52].

Unlike classical computers, that use a bit to represent binary information as a '0' or '1' at a single time, quantum computers use the *qubit*, which is the quantum counterpart of a bit, and have the capability to represent the '0' and '1' states simultaneously [38]. This unique ability of qubits can be owed to an exotic property of quantum mechanics known as *superposition*.

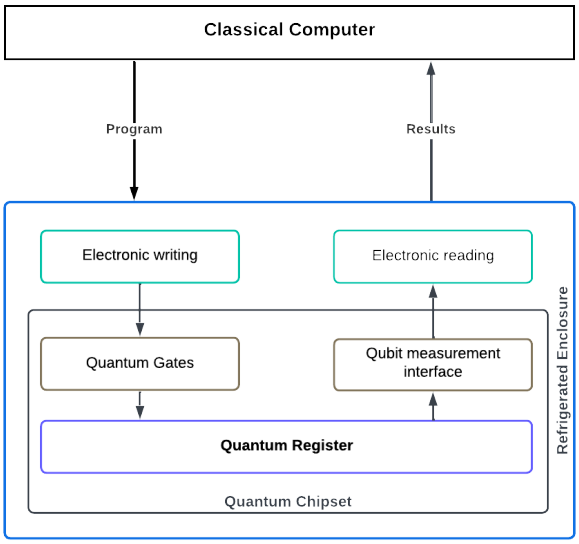


Figure 1.1: Architecture of a Quantum Computer [52]

The theory of superposition is a cornerstone of quantum mechanics, describing a system's ability to be in multiple states simultaneously. For qubits, the quantum analog of classical binary bits, this means they can exist in a state where they are both '0' and '1' at the same time [36]. This phenomenon diverges significantly from classical physics, where objects possess definite properties like position or velocity at any given time [21]. In quantum mechanics, before measurement of a quantum particle's state, particles are not confined to single states but rather exist in all possible states concurrently, as dictated by their wavefunctions [37]. This superposition principle enables quantum systems to occupy many states, which underpins the parallelism that gives quantum computing its potential power [8].

A qubit can exist as the superposition of two fundamental states '0' and '1', while a classical bit cannot. In the realm of quantum mechanics, the states corresponding to '0' and '1' are represented by the two-dimensional vectors $\vec{0}$ and $\vec{1}$, where,

$$\vec{0} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad \text{and} \quad \vec{1} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

The Braket notation, or Dirac notation, is a standard mathematical notation used in quantum mechanics to describe the states of quantum systems, introduced by Paul Adrien Maurice Dirac [15], where a vector $\vec{v}$ is represented as $|v\rangle$. $|\psi\rangle$ is a standard notation used to represent vector state the state of a qubit or quantum system in a complex vector space. For instance, the basic states of a qubit can be represented as $|0\rangle$ and $|1\rangle$, corresponding to the classical states '0' and '1' [36]. Mathematical operations between bras and kets, which calculate the probability amplitude of transitioning from one state to another, are fundamental for predicting outcomes in quantum mechanics [26]. For example, $\langle\psi_1|\psi_2\rangle$ represents the inner product between the vectors $|\psi_1\rangle$ and $|\psi_2\rangle$ respectively. Further in this research, we will utilise the Dirac notation to represent qubit states.

In the context of quantum computing, superposition allows a qubit to be described as a probability amplitude of each of the binary states. A qubit in superposition is mathematically represented as:

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

where, α and β are complex numbers whose absolute squares represent the probability of the qubit being in state $|0\rangle$ or state $|1\rangle$ respectively [36], and these probabilities must sum up to 1, i.e. $|\alpha|^2 + |\beta|^2 = 1$ [38].

The superposition state implies that when a measurement of the qubit's state is made, the qubit "collapses" to one of the basis states, i.e. either $|0\rangle$ or $|1\rangle$. The probabilities of the qubit being in the $|0\rangle$ and $|1\rangle$ states are given by the squared magnitudes of the coefficients α and β in the superposition. This probabilistic nature of quantum states enables

a complex interplay of possibilities of several values at the same time, which is exploited in quantum algorithms to perform calculations that would be impractical on classical computers [43].

For example, Google's Sycamore quantum processor demonstrated quantum supremacy by solving a specific problem known as random circuit sampling[35]. In this problem, the quantum computer generates a sequence of random quantum circuits, and the task is to sample the output distribution of these circuits. Each run of a random quantum circuit on the quantum computer produces a bitstring, i.e. a sequence of binary digits, which, due to quantum interference [48], a phenomenon where the probability amplitudes of different quantum states combine, leading to an enhancement or cancellation of certain outcomes, results in some bitstring sequences more likely to be generated than others. The challenge for classical computers is to simulate this output distribution, which becomes exponentially more difficult as the number of qubits and gate operations increases. Google's Sycamore processor completed this task in 200 seconds, a feat that would take the most advanced classical supercomputers approximately 10,000 years to achieve [33], thereby showcasing the immense potential of quantum computing for solving complex computational problems beyond the reach of classical methods, thus achieving quantum supremacy over classical computers [9].

There are several ways to realise a qubit, the most popular of which is using an electron in the ground state or lowest energy state to represent the $|0\rangle$ state, the electron in the lower energy or $|0\rangle$ state can be moved to the higher energy or $|1\rangle$ state or even be moved to a superposition state of $|0\rangle$ and $|1\rangle$, which can be done by adjusting the duration of time for which light is projected on an atom so as excite the electron and change it's energy state [38]. A qubit can be geometrically visualised in terms of a *Bloch Sphere* [20] representation as given in Figure 1.2.2 below:

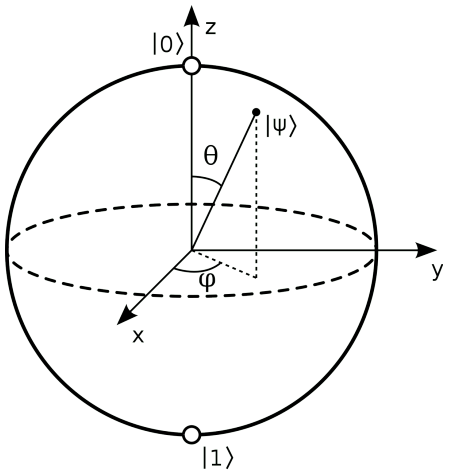


Figure 1.2: Bloch Sphere representation of a qubit

A qubit state can be represented by any point on the surface of the Bloch sphere. Hence, any generalized state $|\Psi\rangle$ of the qubit can be represented by two parameters θ and ϕ as shown below:

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

where, $|cos\frac{\theta}{2}|^2 + |sin\frac{\theta}{2}e^{i\phi}|^2 = 1$ as they are complex numbers, and represent probabilities of the qubit being in the $|0\rangle$ or $|1\rangle$ state respectively.

Quantum computing employs the principles of quantum mechanics, such as superposition and entanglement, enabling it to perform certain computations at speeds unattainable by classical computers, and have ignited intense research into quantum algorithms [17, 47, 52, 53], seeking to exploit quantum computational supremacy to accelerate traditional machine learning (ML) algorithms and pioneer new quantum-based models, i.e. the ability of quantum computer's to execute algorithms so complex or advanced that they that are practically impossible for classical computers to perform in a reasonable amount of time.

This new paradigm of computation not only challenges our longstanding approaches to complex problem-solving but also enhances them, particularly in fields where conventional methods falter under the sheer scale and complexity of data involved. Quantum computers, inherently capable of handling exponentially large datasets through quantum parallelism, introduces a new frontier for machine learning (ML), termed *Quantum Machine Learning (QML).* The intersection of these two disciplines is ripe for exploration, promising breakthroughs in speed, efficiency, and capabilities that could reshape our approach to data science, optimisation problems, and beyond [28].

QML seeks to harness quantum computational supremacy to accelerate classical ML algorithms and develop new quantum algorithms that can outperform their classical counterparts. [32] emphasizes that QML algorithms, like quantum algorithmic equivalents of K-Nearest Neighbours and Support Vector Machine exhibit significant potential in efficiently solving problems involving complex data structures and distance calculations [11]. Results from [11] prove how hybrid models, i.e. a combination of quantum and classical elements, enable quantum computers to significantly surpass classical models in speed and accuracy across various domains, such as finance and healthcare. In addition, [51] advocates for quantum algorithms in classical data processing to improve clustering and classification speed. Studies like [28] demonstrate the practical applications of QML, where quantum and classical ML models are employed together to classify medical data, showing enhanced performance in the classification of non-small-cell lung cancer sub-types [28]. This fusion of quantum and classical methods captures complex probability distributions more effectively, a significant advantage in fields like oncology, where precise classification directly influences treatment outcomes [28].

The integration of classical and quantum computing reflects a growing trend toward hybrid models, which combine the robustness of classical algorithms with the computational speed of quantum processes. For instance, [11] illustrates a hybrid classical-quantum approach for multi-class classification, revealing that quantum annealing, a method that uses principles of quantum mechanics to find the best solution to complex optimisation problems [18] can significantly optimise feature selection and classification tasks in datasets . In the finance sector, [4] highlights how QML surpasses classical models in handling high-dimensional data spaces and complex optimisation landscapes [28]. These hybrid approaches are further supported by recent research by [10], which benchmarks quantum ML models against classical standards and underscores the subtle advantages of quantum-based learning for parallelisable tasks .

However, the transition to quantum-enhanced ML is not without challenges. *Noisy Intermediate-Scale Quantum (NISQ)* [12] devices refer to a class of quantum computers that are characterised by having a moderate number of qubits, typically ranging from 50 to a few hundred, and operate with a certain level of noise and error in their computations and lack sufficient error correction capabilities, which constrains their reliability for performing long or complex quantum algorithms. The current era of NISQ devices [41] presents limitations such as error rates and coherence times that may impede the practical deployment of QML algorithms . Despite these hurdles, the theoretical and experimental progress, such as the development of quantum Boltzmann machines [5], which is a type of stochastic neural network that is based on principles of statistical mechanics [27], for complex biological datasets, suggests a bright future for this technology. Additionally, research by [47] and [34] provides a thorough overview of the landscape, suggesting that Grover's Search, a quantum search algorithm that provides a way to speed up the search for a specific item within an unsorted database [16], and tensor networks could have promising implications for supervised and unsupervised learning models.

Empirical evidence also points toward significant improvements when quantum models are applied to specific instances where classical models struggle. For example, Grover's Search algorithm has shown potential in speeding up the training phase of ML models that deal with large-scale data and offers new ways to approach unsupervised learning and clustering [11]. [42] points out that quantum-enhanced approaches should not be aimed solely at achieving quantum advantage but rather should consider practical relevance and feasibility . The exploration of these advantages in detailed case studies across healthcare and finance underscores the transformative potential of QML [28].

This dissertation aims to provide a high-level analysis of QML's capabilities, comparing them against classical ML algorithms to delineate clear areas of advantage and applicability. Through a synthesis of theoretical insights and empirical studies, it will explore how the unique properties of quantum computing can be leveraged to advance machine learning, potentially leading to a new era of computational intelligence. The following chapters will delve deeper into specific algorithms, hardware considerations, and practical implementations of QML, highlighting its impact and future prospects in the broader landscape of technology and innovation [28].

2 Background Research

2.1 Classical Support Vector Machine (CSVM)

Support Vector Machines are a class of supervised learning algorithms used for classification, regression, and outlier detection. The fundamental objective of an SVM is to find the optimal hyperplane that distinctly classifies data points in an n-dimensional space, where n represents the number of features. This method is particularly effective in high-dimensional spaces and in scenarios where the number of dimensions exceeds the number of samples [44].

The simplest form of SVM is the linear SVM, which aims to find a linear separator (hyperplane) between two classes, as visually depicted in Figure 2.1.

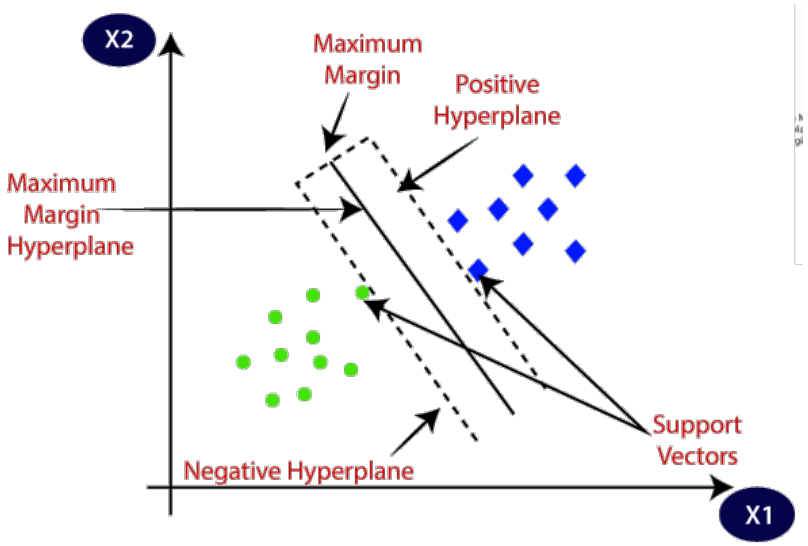


Figure 2.1: Support vectors are data-points that are closest to the maximum margin and lie on the negative and positive hyperplanes, thus orienting it in a N-dimensional space [6]

For a binary classification problem with data points $(\mathbf{x}_i, y_i)$, where $\mathbf{x}_i \in \mathbb{R}^n$ and $y_i \in \{-1, 1\}$, the goal is to find a hyperplane defined by

$$\mathbf{w} \cdot \mathbf{x} + b = 0 \tag{3}$$

where $\mathbf{w}$ is the weight vector and b is the bias term. The decision function for a new data point $\mathbf{x}$ is given by:

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b \tag{4}$$

The primary objective is to maximise the margin, which is the distance between the hyperplane and the nearest data points from both classes, known as support vectors.

To find the optimal hyperplane, we solve the following optimisation problem:

$$\min_{\mathbf{w},b} \frac{1}{2} \|\mathbf{w}\|^2 \tag{5}$$

subject to the constraints: $y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 \quad \forall i$

This is a convex quadratic optimisation problem, which can be solved using techniques such as Lagrange multipliers [44].

The Kernel Trick: For non-linearly separable data, SVM maps the input data into a higher-dimensional space where a linear separator can be found. The kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$ computes the inner product between two data-points $\mathbf{x}_i$ and $\mathbf{x}_j$ in this higher-dimensional space without explicitly transforming the data. Common kernels include the polynomial kernel,

$$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j + c)^d \tag{6}$$

where $\mathbf{x}_i$ and $\mathbf{x}_j$ are the input feature vectors, c is a constant that controls the trade-off between higher-order and lower-order terms in the polynomial, d is the degree of the polynomial, determining the flexibility of the decision boundary [44].

2.2 Quantum Support Vector Machine (QSVM)

QSVMs integrate the principles of quantum computing with classical SVM techniques to leverage the computational advantages of quantum mechanics. This background research outlines the key steps and theoretical concepts involved in QSVM.

2.2.1 ZZ Feature Mapping

The ZZ feature map is used in quantum machine learning to embed classical data into a quantum state space. This embedding allows quantum algorithms to potentially exploit complex patterns and relationships in the data [29].

Qubits: Qubits are the basic units of quantum information, analogous to classical bits but with quantum properties. A qubit can be in a superposition state, meaning it can represent both 0 and 1 simultaneously:

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

where α and β are complex numbers such that $|\alpha|^2 + |\beta|^2 = 1$ [38].

Quantum Gates: These are operations that change the state of qubits [49]. Common gates include:

- **Rotation Gate $R_z(\theta)$:** Rotates a qubit around the z-axis by an angle θ:
$$R_z(\theta) = \begin{pmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{pmatrix}$$
- **Entangling Gate (ZZ Interaction):** Applies a phase based on the product of the states of two qubits:
$$ZZ(\gamma) = e^{-i\gamma Z \otimes Z}$$
where Z is the Pauli-Z matrix, which equals to $\left(\begin{smallmatrix} 1 & 0 \\ 0 & -1 \end{smallmatrix}\right)$, and $Z \otimes Z$ denotes the tensor product of Z with itself. The tensor product works by multiplying every element of one vector, matrix, or tensor with every element of another, creating a new, larger object where each entry is the product of elements from the original inputs, arranged in a structured way, for eg. If A is an $n \times m$ matrix and B is a $p \times q$ matrix, their tensor product $A \otimes B$ is an $np \times mq$ matrix. When this gate applied to a qubit in the $|0\rangle$ state leaves that state unchanged, but transforms a qubit in the $|1\rangle$ state to $-|1\rangle$.

ZZ Feature Map algorithm [29, 45, 49]:

1. **Initial quantum state**

The initial quantum state is $|0\rangle^{\otimes n}$, where n is the number of qubits, where all qubits are in the $|0\rangle$ state.

2. **Applying single-qubit rotations**

For each data point $x = (x_1, x_2, \ldots, x_n)$, a a specific rotation gate $R_z(2\pi x_i)$ is applied to each i-th qubit qubit as the primary goal of the feature map is to encode classical data points into quantum states in a meaningful way, and by applying this specific rotation, we are directly mapping each data point component x_i into the phase of the qubit's state.

Mathematically: $R_z(\theta) = \begin{pmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{pmatrix}$ therefore, $R_z(2\pi x_i)|0\rangle = \begin{pmatrix} e^{-i\pi x_i} & 0 \\ 0 & e^{i\pi x_i} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} e^{-i\pi/2} \\ 0 \end{pmatrix}$.

If x_i is 0.5, for example, the rotation would be: $R_z(\pi)|0\rangle = \begin{pmatrix} e^{-i\pi/2} & 0 \\ 0 & e^{i\pi/2} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} e^{-i\pi/2} \\ 0 \end{pmatrix} = \begin{pmatrix} -i \\ 0 \end{pmatrix}$

3. **Apply entangling gates**

After applying the single-qubit rotations, the ZZ gate is used to entangle the qubits, wherein an operation on one qubit will affect the other qubit(s) it has been entangled with. The ZZ entanglement for two qubits is:

$$ZZ(\gamma) = e^{-i\gamma Z \otimes Z}$$

where, γ is a parameter that controls the strength of the interaction between the two qubits by determining the phase shift applied due to the ZZ interaction, for example, if $\gamma = \pi/2$, the ZZ interaction matrix is:

$$ZZ(\pi/2) = e^{-i(\pi/2)Z \otimes Z} = \begin{pmatrix} e^{-i\pi/2} & 0 & 0 & 0 \\ 0 & e^{i\pi/2} & 0 & 0 \\ 0 & 0 & e^{i\pi/2} & 0 \\ 0 & 0 & 0 & e^{-i\pi/2} \end{pmatrix}$$

This step creates an entangled state, linking the qubits in a way that their individual states cannot be described independently.

Example of the ZZ Feature Mapping [29, 49]

Let's consider a simple example with two data points $x = (x_1, x_2)$ and $y = (y_1, y_2)$:

1. **Initialising the state with 2 qubits:** $|0\rangle = \binom{1}{0}$ and $|0\rangle = \binom{1}{0}$

2. **Applying single-qubit rotations:**

$$|\psi(x)\rangle = R_z(2\pi x_1)|0\rangle \otimes R_z(2\pi x_2)|0\rangle$$

For $x_1 = 0.5$ and $x_2 = 0.3$: $|\psi(x)\rangle = R_z(\pi)|0\rangle \otimes R_z(0.6\pi)|0\rangle$

3. **Applying ZZ entangling gates:**

$$|\psi(x)\rangle = ZZ(\gamma)(R_z(2\pi x_1)|0\rangle \otimes R_z(2\pi x_2))|0\rangle$$

If $\gamma = \pi/2$: $|\psi(x)\rangle = e^{-i(\pi/2)Z\otimes Z}(e^{-i\pi/2}|0\rangle \otimes e^{-i0.3\pi}|0\rangle)$. See section 6.4 for a working ZZ Feature Map circuit with example.

2.2.2 Optimisation Problem in QSVM

The optimisation problem aims to find the optimal hyperplane that separates data points of two classes by maximising the margin and minimising classification errors. The QSVM optimisation problem is formulated similarly to classical SVM but incorporates the quantum kernel function $K_Q(\mathbf{x}_i, \mathbf{x}_j) = |\langle\psi(\mathbf{x}_i)|\psi(\mathbf{x}_j)\rangle|^2$, which is the dot product of vectors $|\psi(\mathbf{x}_i)\rangle$ and $|\psi(\mathbf{x}_j)\rangle$ [30]. The dual form of the optimisation problem in QSVM is:

$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K_Q(\mathbf{x}_i, \mathbf{x}_j) \tag{7}$$

subject to: $\sum_{i=1}^{n} \alpha_i y_i = 0 \quad \text{and} \quad 0 \leq \alpha_i \leq C$

Here, α_i are the Lagrange multipliers, which determine the support vectors that define the optimal hyperplane. Non-zero α_i values correspond to support vectors, while zero values indicate non-support vectors. The constraint $\sum_{i=1}^{n} \alpha_i y_i = 0$ ensures the balance of the hyperplane equation, while $0 \leq \alpha_i \leq C$ introduces a regularisation effect to prevent over-fitting. The regularisation parameter C controls the trade-off between maximising the margin and minimising classification errors, ensuring a balance between bias and variance in the model. This structured approach allows QSVM to effectively classify non-linearly separable data in a high-dimensional feature space, leveraging quantum computing's efficiency and capability to handle complex datasets [30, 32].

2.2.3 Decision Function

The decision function is fundamental for classifying new data points. It relies on the results from the optimisation problem, which identifies the most critical training data points, known as support vectors. These support vectors are characterized by non-zero Lagrange multipliers (α_i) and lie closest to the decision boundary, thus defining the hyperplane that separates different classes [30].

For a given new data point $\mathbf{x}$, the decision function is formulated as:

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i y_i K_Q(\mathbf{x}_i, \mathbf{x}) + b \tag{8}$$

Here, α_i are the Lagrange multipliers from the training phase, y_i are the class labels of the support vectors, $K_Q(\mathbf{x}_i, \mathbf{x})$ is the quantum kernel function measuring the similarity between the training data points $\mathbf{x}_i$ and the new data point $\mathbf{x}$, and b is the bias term.

The decision function works by summing the weighted similarities between the new data point and the support vectors. The weights are determined by the product of the Lagrange multipliers and the class labels of the support vectors. The quantum kernel function projects the data into a high-dimensional quantum feature space, allowing the QSVM to handle non-linearly separable data effectively. The bias term b adjusts the hyperplane's position to accurately reflect the data's distribution [13, 30, 40].

The effectiveness of the decision function comes from its reliance on the support vectors, which are the most informative data points. By focusing on these critical points, the QSVM can efficiently classify new data points. The sign of the

decision function $f(\mathbf{x})$ determines the class of the new data point: if $f(\mathbf{x}) > 0$, the data point is classified into one class (e.g., +1); if $f(\mathbf{x}) < 0$, it is classified into the other class (e.g., -1). This structured approach ensures that the QSVM can robustly and accurately classify complex, high-dimensional data [30, 32, 44].

2.3 Differences Between CSVM and QSVM

The primary difference between classical SVM and QSVM lies in the kernel function. Classical SVM uses predefined kernels such as linear, polynomial, and RBF, whereas QSVM employs a quantum kernel that measures the overlap between quantum states, allowing for more complex and efficient computations [4]. In classical SVM, data is mapped into a higher-dimensional space using kernel functions, while in QSVM, data is encoded into quantum states through quantum feature mapping, representing data in a potentially exponentially large vector space, also known as a Hilbert space [14].

Computational efficiency is another significant difference. QSVM exploits quantum parallelism and entanglement, potentially offering substantial speedups for certain computations compared to classical SVM which may require multiple CPU cores for classical parallelism, especially in high-dimensional spaces [4, 31]. The inner product calculation is also distinct; in classical SVM, it is computed in a transformed feature space defined by the kernel, whereas in QSVM, it is computed between quantum states, providing a novel way to capture data similarities [30].

In summary, Support Vector Machines (SVM) are powerful tools for classification and regression tasks, utilizing margin maximization and the kernel trick to handle non-linearly separable data [1, 13, 46]. Quantum Support Vector Machines (QSVM) enhance the classical SVM framework by utilising quantum computing principles, offering potential computational advantages and the ability to handle complex, high-dimensional data more efficiently [30]. The key differences lie in the kernel function, feature mapping, computational efficiency, and inner product calculation, making QSVM a promising advancement in the field of machine learning.

Quantum Support Vector Machines (QSVM) extend classical SVM algorithms into the quantum domain, leveraging quantum mechanics to enhance classification tasks. This process, described by [24], involves encoding classical data into quantum states using parameterised quantum circuits, facilitating data separation in complex, high-dimensional spaces.

[4] explored the application of QSVM in the financial sector, demonstrating its superiority in handling high-dimensional datasets. They found that QSVMs could process data more efficiently and provide more accurate predictions than classical SVMs. The quantum kernel's ability to represent complex data relationships more effectively than classical kernels is a significant advantage, as noted by [32].

[32] also emphasised the potential of quantum kernels to improve classification accuracy, suggesting that the design of appropriate quantum kernels is crucial for achieving optimal performance in QSVM. Their research highlighted the importance of selecting suitable quantum circuits for specific datasets, as the performance of QSVM can be significantly influenced by the choice of the quantum feature map.

3 Methodology

The hardware and software specifications for this methodology is given in the appendix section 6.2. The practical implementation was executed out in an iPython Notebook via the Jupyter extension in Visual Studio Code. The data pre-processing, CSVM pipeline and QSVM pipeline were implemented in three separate iPython notebooks.

3.1 Data Collection and Initial Inspection

The dataset utilised in this analysis was sourced from the UCI Machine Learning Repository and pertains to breast cancer diagnostics. This dataset was created by Dr. William H. Wolberg, a physician at the University of Wisconsin Hospital in Madison, Wisconsin, USA [50]. The data comprises 569 entries with 33 columns, representing various features extracted from digitized images of fine needle aspirate (FNA) of breast masses. These features describe the characteristics of the cell nuclei present in the images. For this project, the Python programming language was used within a Jupyter Notebook environment. Essential libraries such as *pandas* for data manipulation, *numpy* for statistical analysis, *matplotlib* for data visualisation, and *seaborn* for statistical data visualisation were employed.

Initially, the dataset was imported into a pandas DataFrame for manipulation and analysis via the `pd.read_csv()` function. The dataset contained an 'id' column and an 'Unnamed:32' column with missing values. These columns were removed using pandas' `drop()` function to focus on the significant features. Furthermore, the target variable 'diagnosis'

was converted from categorical ('M' for malignant and 'B' for benign) to numerical (1 for malignant and 0 for benign) using pandas' `map()` function to facilitate analysis.

The structure and summary statistics of the dataset were reviewed using pandas' `info()` and `describe()` functions. These functions helped to understand the distribution and central tendencies of the features, revealing that the dataset was complete with no missing values, and the features varied widely in their range and distribution. Summary statistics provided insights into the means, standard deviations, and percentiles of each feature, aiding in identifying potential outliers and the overall spread of the data.

3.2 Feature Selection - Random Forest (RF) and Correlation Plot

Given the high dimensionality of the dataset (30 features), feature selection was performed using a Random Forest classifier, and by computing computing the correlation of each feature to the 'diagnosis' column to identify the most important features. Random Forests, implemented via the `RandomForestClassifier()` in the scikit-learn library, are ensemble learning methods that build multiple decision trees and merge their results to improve predictive accuracy and control overfitting. A by-product of Random Forests is the feature importance scores, which indicate the contribution of each feature to the prediction.

The Random Forest classifier was trained on the dataset using the `fit()` method, and feature importances were extracted using the `feature_importances_` attribute. Features were ranked based on their importance scores, and the top features were selected for further analysis. A bar plot was generated using marplotlib's `bar()` function to visualise the feature importances. This justified their selection for subsequent analysis and model building.

For correlation analysis, the `corr()` method computed pairwise correlations of columns, creating the correlation matrix for the dataset. The `sort_values()` method was then used to sort the correlation values in descending order, identifying the most correlated features with the target variable.

3.3 Data Wrangling

Data wrangling involved creating a function, `n_feature_wrangling()`, to select a specified number of top features based on their RF feature importance and correlation rankings. This function used the identified top features to subset the original dataset into divisions of 5, 10 and 15 of the most important features, ensuring that only the most relevant features were included. This reduction enhances the efficiency and accuracy of subsequent machine learning models, leading to robust and interpret-able results for breast cancer diagnosis. The processed data was then saved using `to_csv()`, which writes the DataFrame to a CSV file, enabling further analysis and modelling.

3.4 Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) was conducted to uncover patterns and relationships within the data and to visualise the distribution of the target variable and did not go too deep as this is a popular and well-explored dataset in the data science ecommunity. A count plot was generated using seaborn's countplot() function to visualise the distribution of benign and malignant diagnoses. This plot revealed a noticeable imbalance in the dataset, with a higher number of benign cases, i.e. 357 benign compared to 212 malignant ones. Following that, the mean values for each of the features was computed using pandas' `groupby()` function for benign and malignant cases.

A pair plot was created using seaborn's `pairplot()` function to visualise relationships between selected features and the diagnosis variable. This plot was instrumental in identifying potential separability of the classes based on combinations of features. This plot is available as Figure 6.5 in the appendix.

3.5 Classical Support Vector Machine (CSVM)

The CSVM pipeline is initiated by importing in the required libraries and creating a function that easily loads the three variations of the processed data created earlier, i.e. the 5, 10, 15 feature variations of Breast Cancer, and splits it into independent features (X) and the dependent feature (y) which is the 'diagnosis' column. 80% of X and y is further split into training (X_train, y_train; 455 rows/records) and 20% into testing (X_test, y_test; 114 rows/records). Following this, the dimensions of the resultant data splits are inspected to ensure dimensional correctness. This separation is essential for supervised learning, as it distinguishes the input features from the output labels that the model will learn to predict.

Following data importation and splitting, a common function (`svm_train_predict()`) is written that accepts the training and testing splits of X and y, creates a SVM model instance with its' default parameters using the `SVC()`

function of scikit-learn, trains the model instance on X_train and y_train using the `fit()` method, performs predictions using the `predict()` method on X_test and computes accuracy metrics via the `classification_report()` function of scikit-learn of the resultant predictions with y_test on the testing split, that includes precision, recall, f1-score and overall accuracy. Apart from these accuracy metrics, a confusion matrix heatmap is rendered using seaborn's `heatmap()` function, and the `precision_recall_curve()` was used to acquire the precision and recall separately and an area-under-precision-recall-curve (AUPRC) plot is rendered as well using matplotlib's `plot()` function. The time taken for training and predicting was also recorded using the `time()` function. Following this, the 3 variations of the data were passed to this function, and accuracy metrics were computed alongside the time taken for training and testing.

To enhance the model's performance, the training and test data undergo min-max scaling [3]. The min-max scaling formula [25] is given by:

$$X_{\text{scaled}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}$$

where: X is the original data-point being scaled, X_{min} is the minimum value in the entire dataset, X_{max} is the maximum value in the entire dataset, X_{scaled} is the scaled value, which will be in the range (0, 1).

This scaling process normalised the feature values, bringing them within a specified range of 0 to 1, which is beneficial for the SVC model as it relies on distance-based calculations. The scaled training data (X_train_scaled) and scaled test data (X_test_scaled) are then obtained and passed to the `svm_train_predict()` function and the same accuracy metrics alongside training and prediction time was obtained for the scaled data.

3.6 Quantum Support Vector Machine (QSVM)

The start was similar to CSVM's methodology, where a common function was used to load all variations of the dataset, split them into X and y, and then training and testing, including a user-desired option to return the original or min-max scaled version of the data. The data format for Qiskit's `QSVM()` function requires a python dictionary consisting of the combined X_train and y_train training data when fitting the model, and another dictionary containing combined X_test and y_test testing data when performing predictions. The keys correspond to the unique labels of the target feature or 'y' data, in our data's case, there's just two labels (0-benign and 1-malignant), and the values are the individual records in X_train and X_test in the form of a $m \times n$ matrix, where m is the number of rows/records and n is the number of features. Another requirement for the data format is to map the keys to alphabetical strings, so in this case, 0 and 1 is mapped to 'A' and 'B' respectively using the `map()` function. The original and scaled version of the data is acquired in the required format via this function.

A key step in QSVM is mapping classical data into a quantum feature space, and this was using the `ZZFeatureMap()` function of Qiskit, which encodes data points into quantum states using entangling gates and rotations parameterised by the data features, with the `reps` parameter determining the number of repetitions to enhance the expressiveness of the quantum feature space, and the `entanglement` parameter set to 'linear', meaning every qubit is entangled with the adjacent qubit. A custom function named `zz_feature_mapping()` returned the feature mapping of all the three variants of the data with ease.

Following this, another function named `qsvm_train_test()` was used to train and test a QSVM model. The function first created a QSVM model instance using the `QSVM()` function of Qiskit, fetched the backend simulator, i.e. the `statevector_simulator` using the `get_backend()` function of `qiskit.aer`. The Statevector simulator in Qiskit is a tool for simulating quantum algorithms by representing the complete quantum state of the system as a vector. While it offers exact representations of quantum states, its main limitation is scalability, as the state vector's size grows exponentially with the number of qubits. It is particularly useful for debugging and analysing small to medium-sized quantum systems [2].

A quantum instance was then created using the `QuantumInstance()` function, that has the user-defined parameters `backend`, `shots` (number of times the feature map circuit is run through the simulator), `seed_simulator` and `seed_transpiler`, wherein the seed values were set to 42. Model training was initiated using the `run()` method of the QSVM model instance by passing the previously created quantum instance and the training and testing data as parameters. This is where the quantum kernel function is computed in the statevector simulator, this method also performs testing on the model using the test data. Using `time()` function of Python, the total time required for model training is calculated as well, then the total time taken for training and the testing accuracy is printed. Finally the function returns the QSVM model, training time, and the testing accuracy altogether.

A secondary function named `qsvm_predict_accuracy()` performed predictions and computed accuracy metrics for the QSVM model. The predictions were made using the `predict()` by passing the test data (X_test) and the quantum instance holding the statevector simulation backend. The time taken to perform the predictions was recorded as well. Following this, accuracy metrics identical to that in the CSVM pipeline were computed, i.e. precision, recall, f1-score and accuracy using the `classification_report()` function of sklearn, rendering a heatmap of the confusion matrix using seaborn and plotting the AUPRC using the `precision_recall_curve()`, `auc()` functions of scikit-learn and `plot()` function of matplotlib.

The above functions were the backbone of the QSVM pipeline, following this, the 3 data were fed into these functions and used to train three models of the QSVM. The hyperparameter values for the functions are specified in 3.1.

Table 3.1: Hyperparameter Values for Functions in QSVM Pipeline

Data	**zz_feature_mapping()**		**qsvm_train_test()**		
	reps	**entanglement**	**Backend**	**Shots**	**Seed**
5-Feature Variant	4	linear	statevector_simulator	5000	42
10-Feature Variant	5	linear	statevector_simulator	1500	42
15-Feature Variant	1	linear	statevector_simulator	500	42

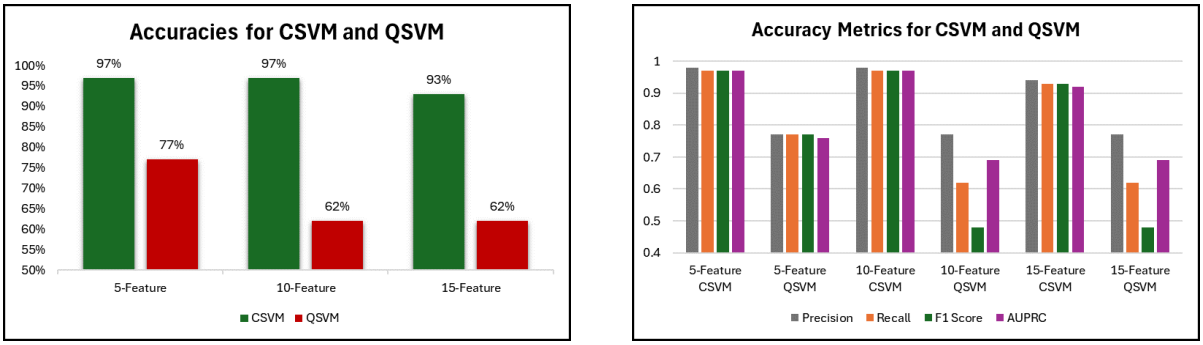
Finally, the model's performance is evaluated using the accuracy metrics specified above The effectiveness of the QSVM comes from its reliance on the support vectors, which are the most informative data points. By focusing on these critical points and leveraging the quantum kernel function, QSVM can efficiently and accurately classify complex, high-dimensional data. Each component, from the data encoding in the feature map to the quantum kernel function and the decision function, plays a crucial role in leveraging quantum computing to enhance the CSVM framework, providing a robust tool for quantum-enhanced machine learning [4, 28].

4 Results

The results recorded throughout the CSVM and QSVM pipelines was training and prediction times, accuracy, precision, recall, f1-score and AUPRC for all three variants of the data [4, 10, 23]. Given in table 4.1 are the tabulated results.

Table 4.1: Comparison of CSVM and QSVM Performance Metrics

Data Type	**Pipeline**	**Train (seconds)**	**Predict (seconds)**	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**AUPRC**
5-Feature	CSVM	0.003	0.001	0.97	0.98	0.97	0.97	0.97
5-Feature	QSVM	14	5.34	0.77	0.77	0.77	0.77	0.76
10-Feature	CSVM	0.004	0.001	0.97	0.98	0.97	0.97	0.97
10-Feature	QSVM	45.9	25	0.62	0.77	0.62	0.48	0.69
15-Feature	CSVM	0.002	0.001	0.93	0.94	0.93	0.93	0.92
15-Feature	QSVM	61.6	30.6	0.62	0.77	0.62	0.48	0.69



(a) Accuracies of CSVM and QSVM models

(b) Precision, Recall, F1-Score, AUPRC metrics

Figure 4.1: Accuracy Metrics for CSVM and QSVM Models

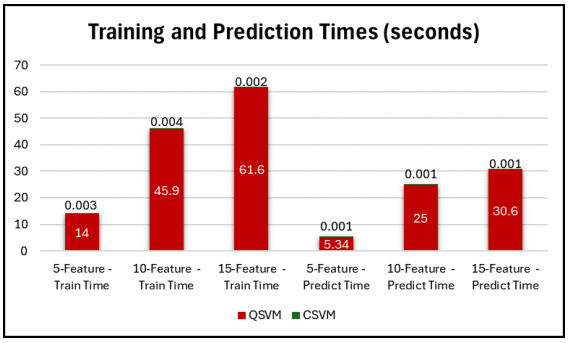


Figure 4.2: Time required for training (first 3 bars) and prediction (last 3 bars) for CSVM (minutely visible) and QSVM

5 Discussion and Conclusion

The results from the comparative analysis between the CSVM and QSVM models across different feature variations reveal several critical insights that are rather counter-intuitive.

Firstly, the training and prediction times for QSVM were markedly higher than those for CSVM across all feature sets. For instance, the training time for the 5-feature QSVM was 14 seconds, compared to a mere 0.003 seconds for the CSVM. Similarly, the prediction times for QSVM were significantly longer. This disparity can primarily be attributed to the use of the statevector simulator, which is implemented on classical hardware. Quantum simulators like the statevector simulator replicate quantum processes using classical computation, which scales exponentially with the number of qubits. This results in a substantial computational overhead, making it impractical for large-scale problems [22].

Secondly, in terms of accuracy, CSVM consistently outperformed QSVM across all feature variations. The accuracy for the 5-feature CSVM was 97%, whereas the QSVM only achieved 67%. Similar trends were observed for the 10-feature and 15-feature variants. The CSVM's superior performance can be attributed to the maturity and optimisation of classical SVM algorithms, which have been refined over decades [1, 6, 40]. In contrast, quantum machine learning algorithms like QSVM are still in their developmental stages and face significant practical challenges, especially when simulated on classical hardware.

The precision, recall, F1 score, and area under the precision-recall curve (AUPRC) further underscore the limitations of QSVM. For example, the F1 score for the 10-feature CSVM was 0.97, compared to 0.48 for QSVM. These metrics indicate that QSVM, when implemented on classical simulators, falls short in capturing the complex relationships in the data that CSVM handles efficiently [30]. This performance gap highlights the need for more advanced quantum hardware to fully realise the theoretical advantages of quantum machine learning algorithms [41].

Such counter-intuitive results can be owed to the fact that the experimentation was carried out on a quantum simulator that utilised classical hardware to simulate the behaviours of quantum hardware [2], which is not entirely representative and true for the behaviour of a real quantum machine. This experimentation just provided a theoretical insight into the working and potential prospects of QSVM and QML in general.

5.1 Conclusion

The comparative study between CSVM and QSVM demonstrates that the CSVM significantly outperforms the QSVM in terms of computational efficiency and accuracy. The primary factor contributing to the QSVM's under-performance is the use of the statevector simulator on classical hardware, which incurs exponential computational costs with increasing qubits [2]. This finding aligns with the current understanding that while quantum algorithms hold theoretical promise, their practical implementation remains challenging due to hardware limitations [41].

To address these challenges, several avenues for future research are recommended. Firstly, the deployment of QSVM on actual quantum hardware is crucial. Quantum devices, unlike simulators, can exploit quantum mechanical properties such as superposition and entanglement more effectively, potentially offering computational advantages [7]. Secondly, there is a need for algorithmic optimisation within the realm of quantum machine learning. This involves exploring different types of quantum feature maps, entanglement strategies, and optimisation techniques specifically designed for quantum systems. Such advancements could enhance the efficiency and performance of quantum algorithms [24]. Lastly, hybrid classical-quantum approaches should be investigated. These approaches can combine classical pre-processing with quantum computation to leverage the strengths of both paradigms. [11, 19, 22].

References

[1] "A support vector machine-based ensemble algorithm for breast cancer diagnosis". In: *European Journal of Operational Research* 267.2 (2018), pp. 687–699. ISSN: 0377-2217. DOI: 10.1016/j.ejor.2017.12.001.

[2] Jakub Adamski, James P Richings, and Oliver Thomson Brown. "Energy Efficiency of Quantum Statevector Simulation at Scale". In: *Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*. SC-W '23. Denver, CO, USA: Association for Computing Machinery, 2023, pp. 1871–1875. DOI: 10.1145/3624062.3624270.

[3] Md Manjurul Ahsan et al. "Effect of Data Scaling Methods on Machine Learning Algorithms and Model Performance". In: *Technologies* 9.3 (2021). ISSN: 2227-7080. DOI: 10.3390/technologies9030052.

[4] Javier Alcazar, Vicente Leyton-Ortega, and Alejandro Perdomo-Ortiz. *Classical versus Quantum Models in Machine Learning: Insights from a Finance Application*. 2020. arXiv: 1908.10778 [quant-ph].

[5] Mohammad H. Amin et al. "Quantum Boltzmann Machine". In: *Phys. Rev. X* 8 (2 May 2018), p. 021050. DOI: 10.1103/PhysRevX.8.021050.

[6] AnalytiXLabs. *Introduction to SVM – Support Vector Machine Algorithm of Machine Learning*. Last accessed 17 July 2021. 2021. URL: https://www.analytixlabs.co.in/blog/introduction-support-vector-machine-algorithm/.

[7] Frank Arute et al. "Quantum supremacy using a programmable superconducting processor". In: *Nature* 574.7779 (2019), pp. 505–510. DOI: 10.1038/s41586-019-1666-5.

[8] Rami Barends et al. "Superconducting quantum circuits at the surface code threshold for fault tolerance". In: *Nature* 508.7497 (2014), pp. 500–503. DOI: 10.1038/nature13171.

[9] Sergio Boixo and Neill Charles. *The Question of Quantum Supremacy*. Last updated on 23 October 2019. 2019. URL: https://research.google/blog/the-question-of-quantum-supremacy/.

[10] Joseph Bowles, Shahnawaz Ahmed, and Maria Schuld. *Better than classical? The subtle art of benchmarking quantum machine learning models*. 2024. arXiv: 2403.07059 [quant-ph].

[11] Avinash Chalumuri, Raghavendra Kune, and BS Manoj. "A hybrid classical-quantum approach for multi-class classification". In: *Quantum Information Processing* 20.3 (2021), p. 119. DOI: 10.1007/s11128-021-03029-9.

[12] Bin Cheng et al. "Noisy intermediate-scale quantum computers". In: *Frontiers of Physics* 18.2 (2023), p. 21308. DOI: 10.1007/s11467-022-1249-z.

[13] Corinna Cortes and Vladimir Vapnik. "Support-vector networks". In: *Machine learning* 20 (1995), pp. 273–297. DOI: 10.1007/BF00994018.

[14] Lokenath Debnath and Piotr Mikusinski. *Introduction to Hilbert spaces with applications*. Academic press, 2005. URL: https://books.google.co.uk/books?id=tYrG-JlqoQQC&lpg=PR7&ots=9fRRGKSedJ&dq=hilbert%20space%20introduction&lr&pg=PR7#v=onepage&q=hilbert%20space%20introduction&f=false.

[15] Paul Adrien Maurice Dirac. *The principles of quantum mechanics*. 27. Oxford university press, 1981. URL: https://books.google.co.uk/books?id=XehUpGiM6FIC&lpg=PP1&pg=PR3#v=onepage&q&f=false.

[16] Jose P Dumas, Kapil Soni, and Akhtar Rasool. "An introduction to quantum search algorithm and its implementation". In: *Data Management, Analytics and Innovation: Proceedings of ICDMAI 2018, Volume 1*. Springer, 2018, pp. 19–31. DOI: 10.1007/978-981-13-1402-5_2.

[17] Vedran Dunjko, Jacob M. Taylor, and Hans J. Briegel. "Quantum-Enhanced Machine Learning". In: *Phys. Rev. Lett.* 117 (13 Sept. 2016), p. 130501. DOI: 10.1103/PhysRevLett.117.130501.

[18] Diego de Falco and Dario Tamascelli. "An introduction to quantum annealing". In: *RAIRO - Theoretical Informatics and Applications* 45.1 (2011), pp. 99–116. DOI: 10.1051/ita/2011013.

[19] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. "Quantum Random Access Memory". In: *Phys. Rev. Lett.* 100 (16 Apr. 2008), p. 160501. DOI: 10.1103/PhysRevLett.100.160501.

[20] Ian Glendinning. "The bloch sphere". In: *QIA Meeting*. 2005, pp. 3–18. URL: https://www.researchgate.net/publication/268270479_The_Bloch_Sphere.

[21] David J Griffiths and Darrell Schroeter. *Instructor's Solutions Manual: Introduction to Quantum Mechanics*. Pearson education, 2005. URL: https://testbankdeal.com/sample/introduction-to-quantum-mechanics-3rd-edition-griffiths-solutions-manual.pdf.

[22] Gian Giacomo Guerreschi and Mikhail Smelyanskiy. *Practical optimization for hybrid quantum-classical algorithms*. 2017. arXiv: 1701.01450 [quant-ph].

[23] Christopher Havenstein, Damarcus Thomas, and Swami Chandrasekaran. "Comparisons of performance between quantum and classical machine learning". In: *SMU Data Science Review* 1.4 (2018), p. 11. URL: https://scholar.smu.edu/datasciencereview/vol1/iss4/11/.

[24] Vojtěch Havlíček et al. "Supervised learning with quantum-enhanced feature spaces". In: *Nature* 567.7747 (2019), pp. 209–212. DOI: 10.1038/s41586-019-0980-2.

[25] Henderi Henderi, Tri Wahyuningsih, and Efana Rahwanto. "Comparison of Min-Max normalization and Z-Score Normalization in the K-nearest neighbor (kNN) Algorithm to Test the Accuracy of Types of Breast Cancer". In: *International Journal of Informatics and Information Systems* 4.1 (2021), pp. 13–20. ISSN: 2579-7069. DOI: 10.47738/ijiis.v4i1.73.

[26] Jack D Hidary and Jack D Hidary. "Dirac notation". In: *Quantum Computing: An Applied Approach* (2021), pp. 377–381. DOI: 10.1007/978-3-030-83274-2_14.

[27] G. E. Hinton. "Boltzmann machine". In: *Scholarpedia* 2.5 (2007). revision #91076, p. 1668. DOI: 10.4249/scholarpedia.1668.

[28] Siddhant Jain et al. "Quantum and classical machine learning for the classification of non-small-cell lung cancer patients". In: *SN Applied Sciences* 2 (2020), pp. 1–10. DOI: 10.1007/s42452-020-2847-4.

[29] Manuel John et al. "Optimizing Quantum Classification Algorithms on Classical Benchmark Datasets". In: *Entropy* 25.6 (2023). ISSN: 1099-4300. DOI: 10.3390/e25060860.

[30] SS Kavitha and Narasimha Kaulgud. "Quantum machine learning for support vector machine classification". In: *Evolutionary Intelligence* 17.2 (2024), pp. 819–828. DOI: 10.1007/s12065-022-00756-5.

[31] Tariq M. Khan and Antonio Robles-Kelly. "Machine Learning: Quantum vs Classical". In: *IEEE Access* 8 (2020), pp. 219275–219294. DOI: 10.1109/ACCESS.2020.3041719.

[32] Ilya Sinayskiy Maria Schuld and Francesco Petruccione. "An introduction to quantum machine learning". In: *Contemporary Physics* 56.2 (2015), pp. 172–185. DOI: 10.1080/00107514.2014.964942.

[33] John Martinis and Sergio Boixo. *Quantum Supremacy Using a Programmable Superconducting Processor*. Last updated on 23 October 2019. 2019. URL: https://research.google/blog/quantum-supremacy-using-a-programmable-superconducting-processor/.

[34] Yeray Mezquita et al. "A review of k-NN Algorithm Based on Classical and Quantum Machine Learning". In: *Distributed Computing and Artificial Intelligence, Special Sessions, 17th International Conference*. Springer. 2021, pp. 189–198. DOI: 10.1007/978-3-030-53829-3_20.

[35] Sean Mullane. *Sampling random quantum circuits: a pedestrian's guide*. 2020. arXiv: 2007.07872 [quant-ph].

[36] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*. Cambridge university press, 2010. URL: https://books.google.co.uk/books?id=-s4DEy7o-a0C&lpg=PR17&ots=NJ3HbqryVw&dq=Michael%20A%20Nielsen%20and%20Isaac%20L%20Chuang.%20Quantum%20computation%20and%20quantum%20information.%20Cambridge%20university%20press%2C%202010&lr&pg=PA1#v=onepage&q&f=false.

[37] Santanu Pattanayak. *Quantum machine learning with Python: Using Cirq from Google research and IBM Qiskit*. Springer, 2021. DOI: 10.1007/978-1-4842-6522-2.

[38] Santanu Pattanayak and Santanu Pattanayak. "Introduction to quantum computing". In: *Quantum Machine Learning with Python: Using Cirq from Google Research and IBM Qiskit* (2021), pp. 1–43. DOI: 10.1007/978-1-4842-6522-2_1.

[39] Francisco Pires. *World First Room Temperature Quantum Computer Installed in Australia*. Last updated on 3 June 2022. 2022. URL: https://www.tomshardware.com/news/world-first-room-temperature-quantum-computer.

[40] Derek A. Pisner and David M. Schnyer. "Chapter 6 - Support vector machine". In: *Machine Learning*. Ed. by Andrea Mechelli and Sandra Vieira. Academic Press, 2020, pp. 101–121. ISBN: 978-0-12-815739-8. DOI: 10.1016/B978-0-12-815739-8.00006-7.

[41] John Preskill. "Quantum Computing in the NISQ era and beyond". In: *Quantum* 2 (Aug. 2018), p. 79. ISSN: 2521-327X. DOI: 10.22331/q-2018-08-06-79.

[42] Maria Schuld and Nathan Killoran. "Is Quantum Advantage the Right Goal for Quantum Machine Learning?" In: *PRX Quantum* 3 (3 July 2022), p. 030101. DOI: 10.1103/PRXQuantum.3.030101.

[43] Peter W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: *SIAM Review* 41.2 (1999), pp. 303–332. DOI: 10.1137/S0036144598347011.

[44] Shan Suthaharan. "Support Vector Machine". In: *Machine Learning Models and Algorithms for Big Data Classification: Thinking with Examples for Effective Learning*. Boston, MA: Springer US, 2016, pp. 207–235. DOI: 10.1007/978-1-4899-7641-3_9.

[45] Yudai Suzuki et al. “Analysis and synthesis of feature map for kernel-based quantum classifier”. In: *Quantum Machine Intelligence* 2.1 (June 2020). ISSN: 2524-4914. DOI: 10.1007/s42484-020-00020-y.

[46] Jasminka Telalović Hasić and Adna Salković. “Breast Cancer Classification Using Support Vector Machines (SVM)”. In: *Advanced Technologies, Systems, and Applications VIII*. Ed. by Naida Ademović, Jasmin Kevrić, and Zlatan Akšamija. Cham: Springer Nature Switzerland, 2023, pp. 195–205. ISBN: 978-3-031-43056-5. DOI: 10.1007/978-3-031-43056-5_16.

[47] Kyriaki A. Tychola, Theofanis Kalampokas, and George A. Papakostas. “Quantum Machine Learning—An Overview”. In: *Electronics* 12.11 (2023). ISSN: 2079-9292. DOI: 10.3390/electronics12112379.

[48] Niels Walet. “Quantum Mechanics”. In: University of Manchester, 2024. Chap. 13.9, pp. 13.9.1–13.9.3. URL: https://phys.libretexts.org/Bookshelves/Quantum_Mechanics/Quantum_Mechanics_(Walet)/13%3A_Miscellaneous_Quantum_Mechanics_Topics/13.09%3A_Quantum_Interference.

[49] Colin P. Williams. “Quantum Gates”. In: *Explorations in Quantum Computing*. London: Springer London, 2011, pp. 51–122. ISBN: 978-1-84628-887-6. DOI: 10.1007/978-1-84628-887-6_2.

[50] William Wolberg et al. *Breast Cancer Wisconsin (Diagnostic)*. UCI Machine Learning Repository. 1995. DOI: 10.24432/C5DW2B.

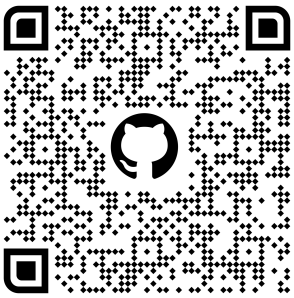
[51] Leonard Wossnig. *Quantum Machine Learning For Classical Data*. 2021. arXiv: 2105.03684 [quant-ph].

[52] Amine Zeguendry, Zahi Jarir, and Mohamed Quafafou. “Quantum Machine Learning: A Review and Case Studies”. In: *Entropy* 25.2 (2023). ISSN: 1099-4300. DOI: 10.3390/e25020287.

[53] Yao Zhang and Qiang Ni. “Recent advances in quantum machine learning”. In: *Quantum Engineering* 2.1 (2020), e34. DOI: 10.1002/que2.34.

6 Appendix

6.1 GitHub Code Repository



— https://github.com/judah-windsor/Battle-of-the-Bit-and-Qubit.git

6.2 Hardware and Software Specifications

1. Processor (CPU): AMD Ryzen 7
2. Memory (RAM): 32 GB DDR5
3. Storage: 2 TB SSD
4. Graphics Processing Unit (GPU): NVIDIA RTX 3060 6 GB
5. Operating System: Windows 11
6. Visual Studio Code: 1.91.1 (user setup)
7. Jupyter extension: v2024.7.0

For CSVM methodology:

1. Python: 3.11.5
2. NumPy: 1.25.2
3. Pandas: 1.5.3

4. Matplotlib: 3.8.2
5. Seaborn: 0.13.2
6. Scikit-learn: 1.5.1

For QSVM methodology (in an anaconda virtual environment):

1. Conda: 23.7.4
2. Python: 3.8.19
3. NumPy: 1.24.4
4. Pandas: 2.0.3
5. Seaborn: 0.13.2
6. Matplotlib: 3.7.2
7. Scikit-learn: 1.3.0
8. Qiskit: 0.32.0
9. Qiskit-aer: 0.9.1
10. Qiskit-aqua: 0.9.5
11. Qiskit-ibmq-provider: 0.18.0
12. Qiskit-ignis: 0.6.0
13. Qiskit-terra: 0.18.3

6.3 Important Features from Random Forest and Correlation Analysis

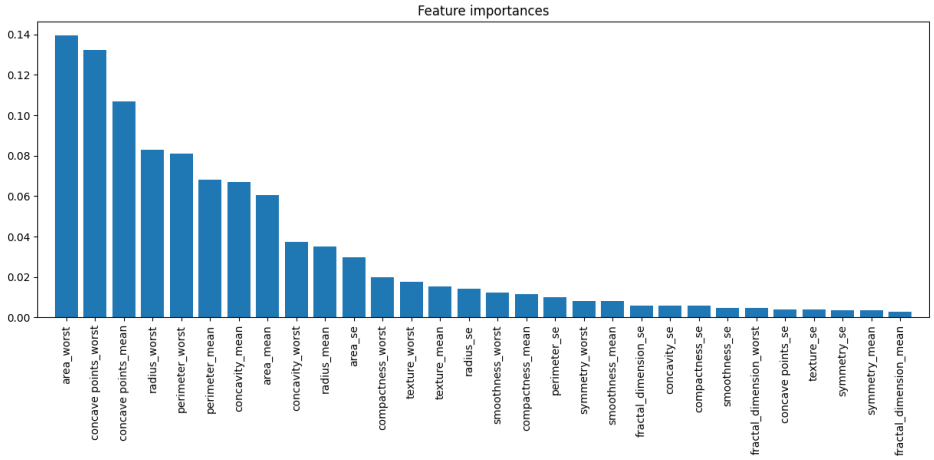


Figure 6.1: Most Important Features - Random Forest

Features selected for 5-feature variant:

1. *concave points_mean*
2. *concave points_worst*
3. *perimeter_mean*
4. *perimeter_worst*
5. *radius_worst*
6. *diagnosis*

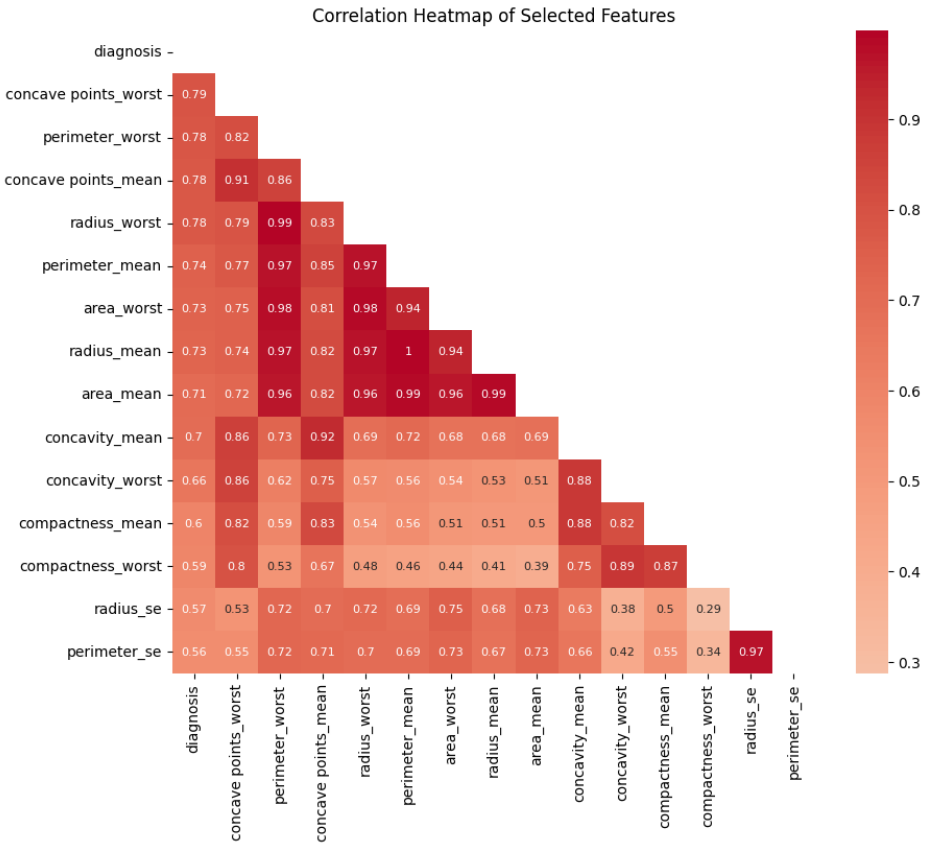


Figure 6.2: 15-Most Important Features - Random Forest

Features selected for 10-feature variant:

1. *area_mean*
2. *area_worst*
3. *concave points_mean*
4. *concave points_worst*
5. *concavity_mean*
6. *concavity_worst*
7. *perimeter_mean*
8. *perimeter_worst*
9. *radius_mean*
10. *radius_worst*
11. *diagnosis*

Features selected for 15-feature variant:

1. *area_mean*
2. *area_se*
3. *area_worst*
4. *compactness_worst*
5. *concave points_mean*
6. *concave points_worst*

7. *concavity_mean*
8. *concavity_worst*
9. *perimeter_mean*
10. *perimeter_worst*
11. *radius_mean*
12. *radius_se*
13. *radius_worst*
14. *texture_worst*
15. *perimeter_se*
16. *diagnosis*

6.4 2-Feature Circuit for ZZ Feature Mapping

The two features selected here are `concave_points_worst` and `perimeter_worst`. Since the number of qubits correspond to number of features, higher number of features result in a more complicated and visually larger circuits which cannot be represented here due to limited resolution, and hence only two features were selected for this explanatory example. Given in figure 6.3 is resultant ZZ Feature Map circuit for the aforementioned features.

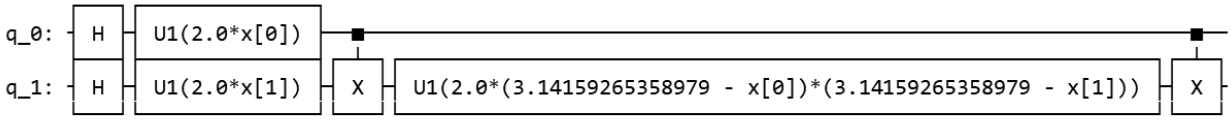


Figure 6.3: ZZ Feature Map Circuit for features `concave_points_worst` and `perimeter_worst`

Here qubits 1 and 2, i.e. `q_0` and `q_1` are assigned to features `concave_points_worst` and `perimeter_worst` respectively, and $\mathbf{x}[0]$, $\mathbf{x}[1]$ are the example data-points. The `H` is the Hadamard gate which sets the qubits in an equal superposition state. After this, the `U1` gate applied a rotation of $2.0 * \mathbf{x}[0]$ radians along the Z-axis of the qubit, when visualised in the bloch sphere. Following this, the `X` gate or popularly known as the `CNOT` gate entangles both qubits, i.e. an operation on one qubit will affect the other in a specified way. Another `U1` gate is applied with a rotation of $2.0 * (\pi - \mathbf{x}[0]) * (\pi - \mathbf{x}[1]$, and this rotation takes effect on both qubits as they are entangled even though it is applied only to `q_2`. This is followed by another `X` or `CNOT` to disentangle the qubits. This is a very primitive example with with 2 features and 2 data points, whilst more complex circuits would involve thousands of such minute operations and qubit rotations.

6.5 Visual insights from the EDA methodology

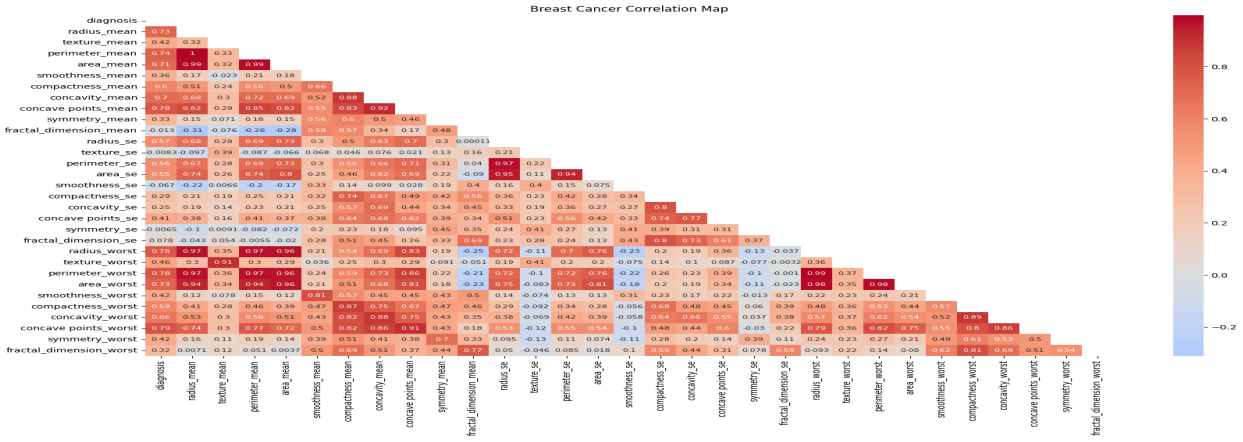


Figure 6.4: Correlation Heatmap of all features in Breast Cancer Data

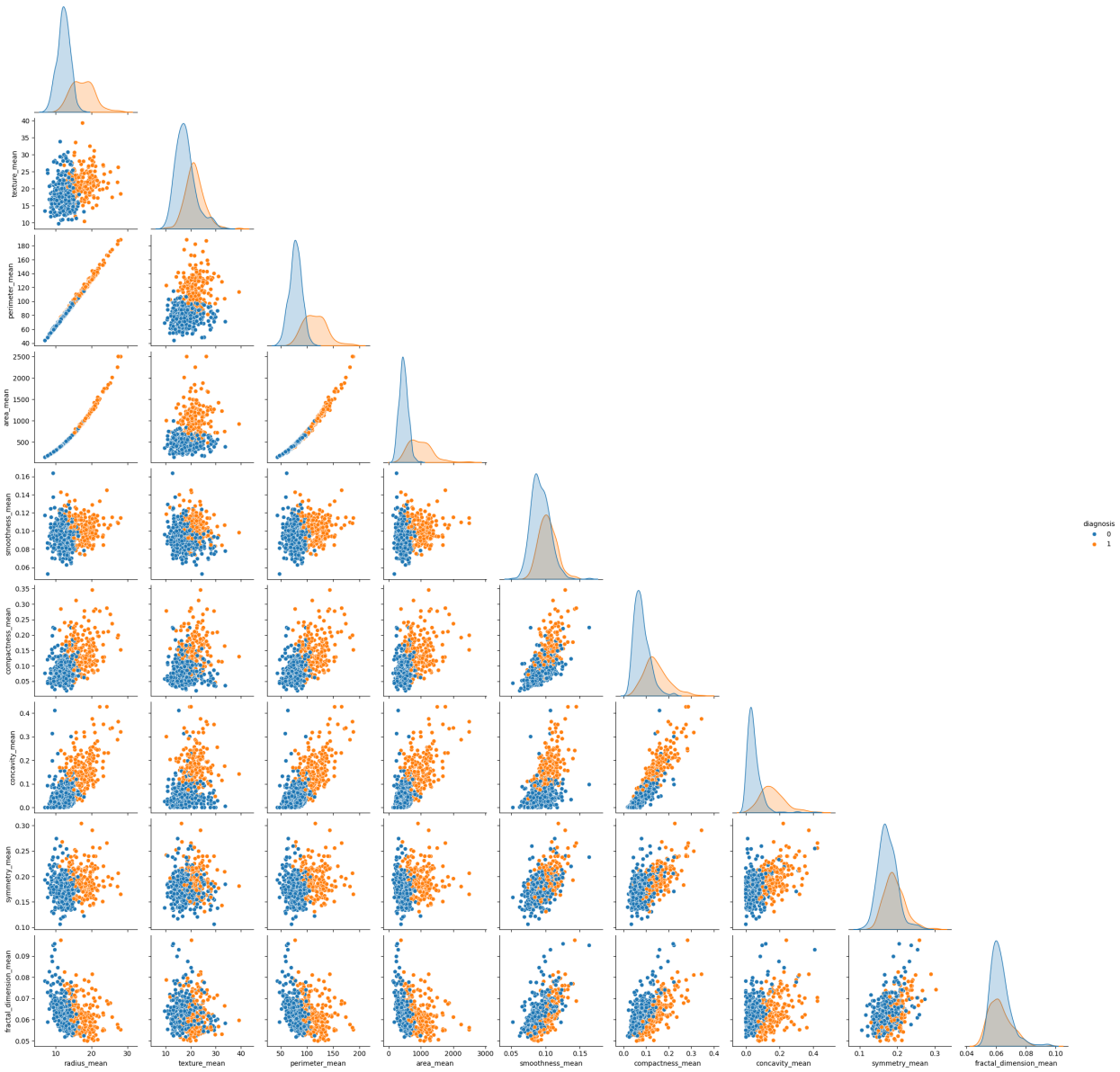


Figure 6.5: Seaborn Pairplot highlighting relationships between different features - Breast Cancer Data. Blue: Benign, Orange: Malignant