uc3m | Universidad Carlos III de Madrid

Master in Statistics for Data Science

2023-2024

Master's Thesis

"Quantum Support Vector Machines: Analysis in Binary and Multiclass Classification"

Iván Torresano Villafranca

Tutors

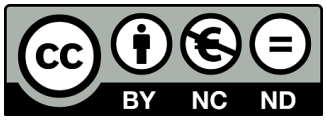
Rosa Elvira Lillo Rodríguez

José Lisandro Aguilar Castro

Madrid, June 25, 2024

DETECCIÓN DEL PLAGIO

La Universidad utiliza el programa **Turnitin Feedback Studio** para comparar la originalidad del trabajo entregado por cada estudiante con millones de recursos electrónicos y detecta aquellas partes del texto copiadas y pegadas. Copiar o plagiar en un TFM es considerado una **Falta Grave**, y puede conllevar la expulsión definitiva de la Universidad.



CONTENTS

1. INTRODUCTION

Machine learning, situated at the crossroads of computer science, statistics, and artificial intelligence, has revolutionized numerous fields by empowering computers to learn from data and make predictions or decisions autonomously, without explicit programming. In today's digital era, its applications in classification are pivotal in sectors including spam mail filters, iris recognition for security systems, customer behavior analysis, and risk assessment in the financial industry. On the other hando, quantum machine learning (QML) integrates principles from quantum mechanics into machine learning algorithms, promising enhanced computational power for solving complex problems. It is expected that in the near future, quantum computers will be able to efficiently solve these and other complex tasks related to processing the growing amounts of global information. Some of the prominent algorithms in QML explored to date include Quantum K-Nearest Neighbors, Quantum Support Vector Machines, Quantum Neural Networks, and Quantum clustering algorithms [2].

Quantum Support Vector Machines (QSVMs) were one of the first machine learning algorithms to be explored in their quantum version for classification tasks. One of the advantages of these models is that they do not require a deep understanding of quantum computing, which facilitates their use by professionals who do not have expertise in quantum computing. These models are based on calculating kernel products using quantum circuits. This study delves into the application of quantum computing principles in QML, focusing on QSVM for binary and multiclass classification tasks. This Master Thesis analyzes their potential to outperform classical SVMs and other commonly used ML algorithms such as Logistic Regression and Random Forest in terms of efficiency and performance on tabular datasets.

In Chapter 2, fundamental principles of Quantum Mechanics are introduced, essential for understanding the basics of Quantum Computing. Chapter 3 explores quantum circuits and their main elements for isolated qubits and multi-qubit systems, which are crucial for grasping the theory underpinning QSVM models. Chapter 4 provides an overview of Quantum Machine Learning types, a detailed development of classical SVMs, the emergence of their quantum counterparts, and a description of the most well-known Quantum Kernels. Next, Chapters 5 and 6 introduce the Case Design structure and include the generation of synthetic datasets with different characteristics. These datasets serve as a foundation for applying QSVM models, classical SVMs with traditional kernels, and other classical Machine Learning algorithms. Their performance and efficiency are compared using metrics such as accuracy, F_1, Mathews Correlation Coefficient (MCC), and by measuring the execution time of each model. Finally, Chapter 7 discusses the results obtained from these comparisons and draws conclusions based on the findings.

2. INTRODUCTION TO QUANTUM MECHANICS

To understand the basic principles of Quantum Computing and Quantum Machine Learning it is necessary to introduce some concepts about Quantum Mechanics, which is a very consolidated theory in physics. Unlike other physical theories, it is not a deterministic one, instead, it is based on computing probabilities

In Quantum Mechanics, the state of a physical system is represented by a "ket" $|\psi\rangle$, which is a vector that belongs to a Vector Space ξ called "Hilbert Space". This notation is called "Dirac Notation" and is widely used in Quantum Mechanics. The dimension of ξ can be finite or infinite, depending on the system. In Quantum Computing, the dimension of the Hilbert Space is always finite, in other words, the kets that represent the quantum states, are vectors of $\mathbb{C}^n$ with $n \in \mathbb{Z}$. Given a ket $|\psi\rangle$, the notation for its dual vector is $\langle\psi|$ and the inner product between 2 kets ψ_1 and ψ_2 is written as $\langle\psi_1|\psi_2\rangle$. Bearing in mind this notation, we will define the postulates of Quantum Mechanics that will be essential to introduce the underlying theory of Quantum Computing.

1. **Postulated I:** The state of a physical system is defined by a ket $|\psi\rangle$ belonging to $\mathbb{C}^n$.

   As a consequence of the properties of Vector Spaces, this principle implies that a linear combination of kets is a valid physical state.

2. **Postulated II:** Every physical quantity is represented by an operator A that acts on $\mathbb{C}^n$ and the only possible values obtained when measuring A are its eigenvalues. Concretely, if there is a physical system with state $|\psi\rangle$ and a physical quantity represented by the operator A, the probability of obtaining a_i when measuring A is:

   $$P(a_i) = |\langle u_i|\psi\rangle|^2 \tag{2.1}$$

   Where $|u_i\rangle$ is the eigenvector of A with eigenvalue a_i. It is useful to stress that a physical state $|\psi\rangle$ can always be expressed as a linear combination of the eigenvectors $|u_i\rangle$ since they form an orthonormal basis in $\mathbb{C}^n$.

   $$|\psi\rangle = \sum_{i=1}^{n} c_i|u_i\rangle \quad with \quad c_i = \langle u_i|\psi\rangle \tag{2.2}$$

3. **Postulated III:** If a measurement of the quantity A on the state $|\psi\rangle$ gives, as a result, a_i, the state ceases being in a superposition of states and instead assumes a single state given by the normalized eigenvector $|u_i\rangle$ associated with a_i. This quantum phenomenon is known as **Wave Function Collapse**

4. **Postulated IV:** As a consequence of postulates 2 and 3, using the Probability properties, all kets that represent quantum states must be normalized:

$$\langle\psi|\psi\rangle = 1 \tag{2.3}$$

A corollary of these results is that matrices A representing operators acting on kets must be unitary. In other words, they must satisfy the following condition:

$$A^\dagger A = AA^\dagger = I \tag{2.4}$$

The reason behind using unitary matrices is that, after applying these transformations, the resulting state $|\psi'\rangle = A|\psi\rangle$ remains normalized.

$$\langle\psi A^\dagger|A\psi\rangle = \langle\psi|A^\dagger A\psi\rangle = \langle\psi|\psi\rangle = 1$$

5. **Postulated V:** The evolution in time of a state $|\psi\rangle$ is given by the popular Shrödinger equation.

$$i\hbar\frac{d|\psi(t)\rangle}{dt} = H|\psi(t)\rangle \tag{2.5}$$

In this equation, H is an operator called 'Hamiltonian'. In physics, the Hamiltonian represents the total energy of a system. The parameter $\hbar$ is a fundamental constant in physics, often referred to as the reduced Planck constant. In this equation, $\hbar$ appears multiplied by i, which denotes the imaginary unit. In quantum computing, although with a different meaning, the Hamiltonian appears in important algorithms such as **Quadratic Unconstrained Binary Optimization(QUBO)** and **Variational Quantum Eigensolver(VQE)**. For this project, there is no need to know how to solve the Shrödinger equation. It is only necessary to know that being a linear differential equation, if there are several solution states $|u_i\rangle$ to the equation, then a new state $\psi = \sum_{i=1}^{n} c_i|u_i\rangle$ which is a linear combination of these states will also be a solution to the Schrödinger equation. This property of quantum systems is known as **Superposition Principle**

2.1. Tensor product and multiple Quantum States

Until now, we have explored the rules that an isolated quantum system $|\psi\rangle$ follows. Nevertheless, there are situations in which there exist several quantum systems $|\psi_i\rangle$, and we want to treat them as a whole system, taking into account the interactions among them. In such situations, the mathematical concept of tensor product arises.

The tensor product $\otimes$ is a well-defined mathematical operation. The notation for the tensor product of 2 kets $|\psi\rangle$ and $|\phi\rangle$ is:

$$c \tag{2.6}$$

Among the most important properties of tensor product are:

1. Linearity for the multiplication by complex numbers

$$|\lambda\phi_1\phi_2\rangle = \lambda|\phi_1\phi_2\rangle \tag{2.7}$$

2. Distributive with respect to the sum of kets

$$|\psi\rangle \otimes (|\phi_1\rangle + |\phi_2\rangle) = |\psi\phi_1\rangle + |\psi\phi_2\rangle \tag{2.8}$$

3. If there are n quantum states $|\phi_i\rangle$, each of them with dimension m_i, the dimension of the ket representing the overall state $|\psi\rangle = |\phi_1\rangle \otimes |\phi_2\rangle \otimes ... \otimes |\phi_n\rangle$ is $m_1 m_2 .. m_n$

When considering quantum systems composed of several isolated systems or 'particles' a new interesting property comes to light. This property is called **Entanglement** and plays a crucial role in Quantum Computing. To understand this property, it is necessary to introduce the term "product states", which is used to describe quantum states formed by taking the tensor product of multiple states. By definition, "product states" are said to be not entangled. Thus, entangled states are all states that cannot be expressed as the tensor product of other states. To see what it implies for a quantum state to be entangled, we will consider a simple case in which there are 2 single systems S_1 and S_2, each of them with dimension 2 and basis kets $|0\rangle$ and $|1\rangle$. Assuming that S is in the the following state, which is not a product space:

$$|\psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \tag{2.9}$$

The possible results obtained when measuring the state of S_1 are $|0\rangle$ and $|1\rangle$, both with probability $p = 1/2$. If we find that S_1 is in the state $|1\rangle$, then, $|\psi\rangle$ will collapse to $|11\rangle$ and hence, the probability that S_2 is in the state $|1\rangle$ is 1, that is, when the state of S_1 is measured, the state of S_2 becomes completely determined.

$$\begin{gathered} |\psi\rangle \xrightarrow{S_1=1} |\widetilde{\psi}\rangle = |11\rangle \\ P(S_2 = 1|S_1 = 1) = |\langle 11|\widetilde{\psi}\rangle|^2 = 1 \end{gathered} \tag{2.10}$$

In essence, it is as if both systems were 'entangled' between them, and after measuring the state of one system, the other 'knows' the result and its state becomes completely determined. Therefore, this phenomenon is called **Entanglement**, and despite how counterintuitive may seem, this quantum property has been experimentally measured plenty of times.

3. QUANTUM COMPUTING

First of all, let's define what is quantum computing. The term "quantum computing" refers to the use of properties of a quantum system such as superposition and entanglement to reduce the computational cost of certain algorithms. In contrast with classical computers, in quantum computers, these properties are explicitly used and as a consequence, there are certain types of algorithms and computational tasks in which they can be useful and outperform classical computers in terms of computational efficiency.

In the paradigm of quantum computing, there are several different models. The most popular one that will be used, is the **quantum model circuit**. As its name suggests, it is based on classical logical circuits and applies the properties of the quantum world.

Figure 3.1
An example of a basic quantum circuit with 2 qubits.

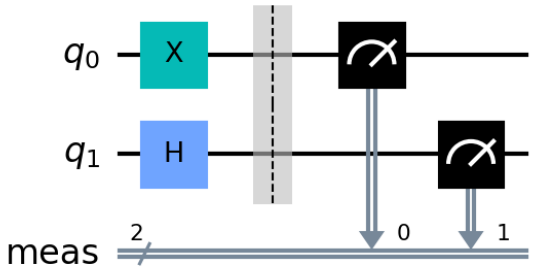


Figure 3.1 shows the schematic representation of a basic quantum circuit, where the letters q_0 and q_1 represent the qubits of the circuit, X and H are the gates applied to each qubit and the black box with a downward arc crossing a line depicts the measurement of a qubit causing its quantum state to collapse yielding a specific measured value or state.

Qubits are the most fundamental unit in quantum circuits. They are analogous to classical bits, which can be in states 0 and 1. In Quantum Computing, the corresponding states are $|0\rangle$ and $|1\rangle$ respectively. In each quantum circuit, the kets are represented by vectors that belong to $\mathbb{C}^n$ where n denotes the number of qubits in that quantum circuit.

3.1. Systems of one qubit

In quantum systems with one qubit, the kets $|0\rangle$ and $|1\rangle$ form an orthonormal basis in $\mathbb{C}^2$. Unlike classical bits, a qubit can exist not only in one of these states but also in a superposition of them due to the Superposition Principle.

Generally, for systems with a single qubit, the expression for its state $|\psi\rangle$ is written as:

$$|\psi\rangle = a\,|0\rangle + b\,|1\rangle \tag{3.1}$$

The parameters a and b are complex numbers called amplitudes and they satisfy the expression $|a|^2 + |b|^2 = 1$ due to the normality condition (2.3). We can compute the probability that when we measure the state of the system, its state collapses to $|0\rangle$ or $|1\rangle$. These probabilities are calculated with the following formulas:

$$P(|\psi\rangle = |0\rangle) = |\langle 0|\psi\rangle|^2 = |a|^2 \qquad P(|\psi\rangle = |1\rangle) = |\langle 1|\psi\rangle|^2 = |b|^2 \tag{3.2}$$

3.1.1. Quantum gates in systems of one qubit

In the quantum circuit model, the operators that act on the qubits are called quantum gates. These gates are represented by unitary matrices, that is, matrices that meet the condition (2.4). For one-qubit systems, as each state corresponds to a vector belonging to $\mathbb{C}^2$, quantum gates are represented by $2x2$ complex matrices. The most important and commonly used quantum gates are:

1. -*NOT* **or** *X* **gate**: It is analogous to the *NOT* gate in classical circuits

$$X|0\rangle = |1\rangle \quad X|1\rangle = |0\rangle \tag{3.3}$$

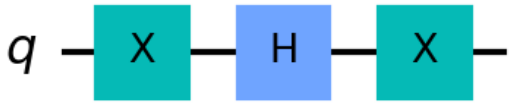
2. - **Hadamard or** *H* **gate**: It has no classical analog and it creates a superposition of quantum states

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \qquad H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \tag{3.4}$$

3. -*Z* **gate**: It is obtained by applying an *H* gate, then an *X* gate and, finally, another *H* gate.

$$Z|0\rangle = |0\rangle \qquad Z|1\rangle = -|1\rangle \tag{3.5}$$

Representation of the *Z* gate applied in a one-qubit system.



The quantum gates mentioned above constitute a specific case of more general gates known as rotation gates. To comprehend rotation gates, it is useful to fact that the state of a one-qubit system can be expressed as:

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle \tag{3.6}$$

The parameters θ and ϕ satisfy that $\theta \in [0, \pi]$ and $\phi \in [0, 2\pi]$ and they can be interpreted as the polar and azimuthal angles of a sphere. Thus, the quantum states of a single qubit can be thought of as points on the surface of a sphere. In this context, it is natural to define the rotation gates $R_X(\theta)$, $R_Y(\theta)$, $R_Z(\theta)$ which apply a rotation of θ degrees around the X,Y and Z axes respectively. It is important to state that for one-qubit systems, any quantum gate U can be expressed as a composition of these rotation gates.

$$U = R_X(\alpha)R_Y(\beta)R_Z(\gamma) \tag{3.7}$$

3.2. Multi-qubit systems

For systems with n qubits, the global state of the system is obtained by 'multiplying' the individual states of each qubit using the tensor product $\otimes$. The simplest and most illustrative case of multi-qubit systems is a quantum circuit with 2 qubits.

$$\begin{aligned} &|\psi\rangle = a_{00}|00\rangle + a_{01}|01\rangle + a_{10}|10\rangle + a_{11}|11\rangle \\ &|a_{00}|^2 + |a_{01}|^2 + |a_{10}|^2 + |a_{11}|^2 = 1 \end{aligned} \tag{3.8}$$

In this case, the kets that represent each possible state, form an orthonormal basis in $\mathbb{C}^4$. From this particular case, it is easy to generalize and obtain an expression for the quantum state of a system with n qubits:

$$\begin{aligned} |\psi\rangle &= \sum_{k=0}^{2^n-1} a_k|k\rangle \\ \sum_{k=0}^{2^n-1} |a_k|^2 &= 1 \end{aligned} \tag{3.9}$$

Where the ket $|0\rangle$ corresponds to all qubits in the state 0, $|1\rangle$ corresponds to the first qubit in the state 1 and the rest in the state 0, and so on. It is key to observe that the number of parameters required to describe the general state of an n-qubit system grows exponentially with n. Part of the strength of quantum computing lies in the ability to implicitly handle 2^n complex numbers by manipulating only n qubits.

3.2.1. Multi-qubit quantum gates

As we have seen above, the state of a system with n qubits is represented by a vector belonging to $\mathbb{R}^{2^n}$. Therefore, the corresponding quantum gates will be $2^n x 2^n$ matrices. If we apply a quantum gate U_i to the i-th qubit of the system, the resulting state will be given by:

$$(U_1 \otimes U_2 \otimes ... \otimes U_n)|\psi_1\psi_2...\psi_n\rangle = |U_1\psi_1 U_2\psi_2...U_n\psi_n\rangle \tag{3.10}$$

If no quantum gate is applied on the i-th qubit, then $U_i = I$. It is worth noting that not all quantum gates within a multi-qubit system can be expressed as the tensor product of one-qubit quantum gates. An illustration of such quantum gates is found in controlled gates. To apply a controlled gate, we must select the qubits in which we want to apply this gate and the target qubit. In this scenario, the quantum gate modifies the state of the target qubit only if the remaining selected qubits are in the state $|1\rangle$. Worthy examples of controlled gates include the $CNOT$ gate for systems of 2 qubits and the $CCNOT$ or Toffoli gate for systems with 3 qubits, both of which execute a controlled-NOT gate on the target qubit.

4. QUANTUM SUPPORT VECTOR MACHINES

4.1. Quantum Machine Learning

Quantum machine learning (QML) involves using quantum computing to enhance classical ML algorithms, in order to improve performance and reduce computational time. QML algorithms can be categorized into four different families, depending on the nature of the data (classical or quantum) and the utilization of quantum computing during model training:

1. **CC:** Both the data and model training are classical, but the algorithms draw inspiration from quantum computing principles.
2. **QC:** The machine learning algorithm is classical, but the data is quantum, obtained by measuring the states of a quantum circuit.
3. **CQ:** The data is classical, but quantum computing is employed in certain steps of the model.
4. **QQ:** Both the data and algorithm are quantum. However, these QML algorithms face challenges due to the current limitations in quantum technology development.

One of the most known CQ quantum machine learning is Quantum Supported Vector Machine (QSVM) which is a classical machine learning algorithm that uses quantum computing for mapping the data into a space of higher dimension (kernel) [1]. Other examples of QC algorithms include Quantum Neural Networks, Hybrid Networks, and Quantum Generative Adversarial Networks [2].

As mentioned, QSVM constitutes a family of QML algorithms that leverage quantum computing for model training. QSVM was among the earliest QML algorithms to be developed and is widely used in binary classification. In the following section, we will explore the underlying theory behind Quantum Supported Vector Machines. First, we will provide a detailed explanation of classical Supported Vector Machines for binary classification as well as its extension for classification tasks with more than 2 classes.

4.2. Support Vector Machines (SVM)

A Support Vector Machine (SVM) is a supervised machine learning algorithm used for classification and regression, although, in this project, we will only focus on its application for classification purposes. Firstly, we will discuss SVM models for binary classification tasks, and then we will extend the discussion to multiclass classification.

In classification tasks, SVM models treat the rows of tabular data as points $\vec{x}$ living in an Euclidean space $\mathbb{R}^p$, where p is the number of features in the dataset. The aim is to find a hyperplane that separates points belonging to different classes. Mathematically, a hyperplane in a euclidean space $\mathbb{R}^p$ can be represented by the equation:

$$\pi : \vec{w}\vec{x} + b = 0 \tag{4.1}$$

Where $\vec{w} \in \mathbb{R}^p$ is a vector orthogonal to the hyperplane and $b \in \mathbb{R}$ is a constant. Bearing (4.1) in mind, it can be distinguished the side on which a point $\vec{y}$ is located with respect to the hyperplane by looking at the sign of $\vec{w}\vec{x} + b$. What an SVM does in practice is to label a row $\vec{x}$ of the dataset depending on which side of the hyperplane is. The adjustable parameters of this model are the components of the normal vector $\vec{w}$ and the parameter b.

4.2.1. Training a linear SVM for binary classification

In a tabular dataset used for training in binary classification tasks, each row represents an observation and is represented by a feature vector $\vec{x_j}$ in $\mathbb{R}^p$, where j denotes the index of the observation. The corresponding label for each observation is denoted as y_j. For convenience in binary classification, the labels of the target variable are encoded as $y_j = -1$ for one class and $y_j = 1$ for the other one. During training, the Support Vector Machine searches for the hyperplane that maximizes the distance to the training points $\vec{x_j}$ while separating points belonging to different classes. The reason behind this is that is expected that points coming from unseen data belonging to one class will be close to the training points of the same class. A very important assumption made in this model is that data can be perfectly separated by a plane, which is known as linear SVM. Nevertheless, this assumption does not always hold since data may not be linearly separable. In these cases, it is defined a function known as kernel to map the data from a space with dimension p to a space of higher dimension, where it is more likely to be linearly separable. Supported Vector Machine models with kernel functions are key in building the corresponding quantum version.

There are two distinct types of models to find the hyperplane that separates the classes depending on the nature of the data: hard-margin and soft-margin.

1. **Hard-margin model**

   It assumes that data points belonging to different classes can be perfectly separable by a hyperplane, which is not a realistic assumption at all but serves as a benchmark to build a model with less restrictive constraints. Considering that there exists one hyperplane that perfectly divides the data, then there exist infinitely many of them. To select the optimal one, a hyperplane H that perfectly separates the data is initially identified. Subsequently, a hyperplane H_+ parallel to H is chosen, which contains the closest point to H, and the reflection of H_+ over H, denoted as H_-. It is essential to note that both hyperplanes H_+ and H_- are parallel to H and equidistant from it. The distance between these auxiliary planes is known as the margin, and the primary objective of the model is to maximize this quantity.

   Equations of the hyperplanes

$$\begin{aligned} H &: \vec{w}\vec{x} + b = 0 \\ H_+ &: \vec{w}\vec{x} + b = C \\ H_- &: \vec{w}\vec{x} + b = -C \end{aligned} \tag{4.2}$$

   Choosing the parameters $\vec{w} = \vec{w}/C$ and $\widetilde{b} = b/C$, the equations corresponding to the hyperplanes (4.2) simplify to:

$$\begin{aligned} H &: \vec{w}\vec{x} + \widetilde{b} = 0 \\ H_+ &: \vec{w}\vec{x} + \widetilde{b} = 1 \\ H_- &: \vec{w}\vec{x} + \widetilde{b} = -1 \end{aligned} \tag{4.3}$$

   For this choice of parameters, the value of the margin is $2/\|\vec{w}\|$. Therefore, the problem of finding the optimal hyperplane for separating the data can be formulated as the following optimization problem:

$$\begin{aligned} &\textit{minimize} \ \frac{\vec{w}^2}{2} \\ &\textit{subject to} \ \ y_j(\vec{w}.\vec{x_j} + \widetilde{b}) \geq 1 \ \forall j = 1, ..., n \end{aligned} \tag{4.4}$$

   Maximizing the value of the margin corresponds to finding the minimum of $\|\vec{w}\|$. As the Euclidean norm $\|\cdot\|$ contains square roots, it is minimized $\vec{w}^2$ instead, which is mathematically simpler. The constraint equations from (4.4), indicate that there cannot be any points $\vec{x_j}$ between H_+ and H_- (inside the margins) and are derived from the hyperplane's equations (4.3).

2. **Soft-margin model**

This model relaxes the assumption of perfect separability between data points belonging to different classes. It allows for points to exist within the margin and for data points of one class to reside on the opposite side of the hyperplane. This relaxation of the constraints is facilitated by incorporating small quantities ϵ_j in equation (3.4), allowing points to reside within a distance ϵ_j inside the margins. To construct an SVM model with strong predictive capability, it is convenient to choose the smallest possible values of ϵ_j. This constraint is integrated as a cost term in the minimization function, penalizing larger values of ϵ_j.

$$\begin{aligned} &minimize \ \frac{\vec{w}^2}{2} + C\sum_{j=1}^{n}\epsilon_j \\ &subject \ to \ y_j(\vec{w}.\vec{x_j} + \widetilde{b}) \geq 1 - \epsilon_j \ \forall j = 1, ..., n \\ &\epsilon_j \geq 0 \end{aligned} \tag{4.5}$$

The constant C is a hyperparameter of the model, and finding its optimal value involves using techniques such as grid-search, random-search, or K-fold cross-validation. When C is set to higher values, it tightens the margin and reduces misclassification. However, excessively high values of this hyperparameter can lead to overfitting, as it makes the model less tolerant of errors and more prone to capturing noise in the data, which is equivalent to the hard-margin case.

The optimization problem of the soft-margin model can be reformulated as an equivalent optimization problem, which is generally, easier to solve:

$$\begin{aligned} &Maximize \ \sum_j \alpha_j - \frac{1}{2}\sum_{j,k} y_j y_k \alpha_j \alpha_k \vec{x_j}.\vec{x_k} \\ &subject \ to \ 0 \leq \alpha_j \leq C \\ &\sum_j y_j \alpha_j = 0 \end{aligned} \tag{4.6}$$

This formulation is known as **Lagrangian dual formulation** and the parameters α_j are known as support vectors and are the reason behind the name of SVM models. Once the optimization problem has been solved, the parameters $\vec{w}$ and b can be derived from the support vectors α_j. Obtaining the vector orthogonal to the hyperplane $\vec{w}$ is straightforward:

$$\vec{w} = \sum_j y_j \alpha_j \vec{x_j} \tag{4.7}$$

This expression is very revealing since it shows that the normal vector $\vec{w}$ only depends on the rows $\vec{x_j}$ with non-zero support vectors.

The parameter b can be obtained by solving the following equation

$$y_s(\vec{w}.\vec{x_s} + b) = 1 \tag{4.8}$$

Obtaining:

$$b = y_s - \sum_j y_j \alpha_j \vec{x_j}.\vec{x_s} \tag{4.9}$$

Where it is satisfied that the support vector corresponding to the index s is strictly positive $\alpha_s > 0$ [3]. Therefore, to assign the class of a new data point $\vec{x}$, it has to be computed the following quantity:

$$\vec{w}.\vec{x} + b = y_s + \sum_j y_j \alpha_j \vec{x_j}.(\vec{x} - \vec{x_s}) \tag{4.10}$$

If the sign of this expression is greater than 0, $\vec{x}$ is classified as positive ($y = 1$); otherwise, it is classified as negative ($y = -1$)

4.2.2. Extension of SVM for multiclass classification

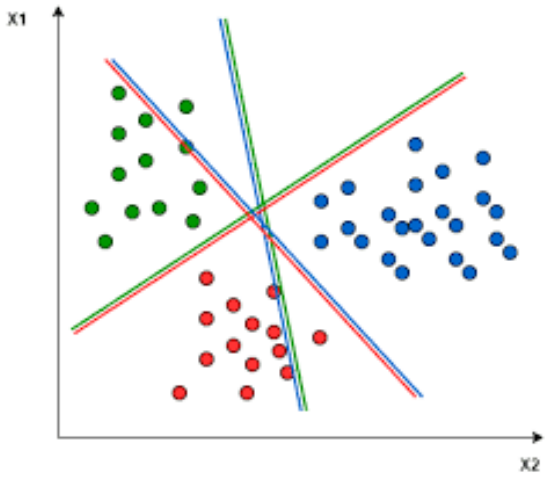
Vanilla SVM models, by default, are designed for regression and binary classification tasks and do not inherently support multiclass classification. However, SVM models can be extended to handle multiclass classification by decomposing the problem into multiple binary classification tasks. This extension of multiclass classification for SVM models can be achieved in two primary ways:

1. **One vs One approach**

   This approach utilizes a binary SVM classifier to separate each pair of classes, neglecting the rest of the points when determining the hyperplane for each pair (Figure 4.1). In this method, if there are m different classes, the model will require training $\frac{m(m-1)}{2}$ different classifiers.

   Consequently, the complexity and computational time of the model grow quadratically with the number of classes m.

Figure 4.1
Graphical representation of the hyperplanes separating each pair of classes in the One vs One approach [9]

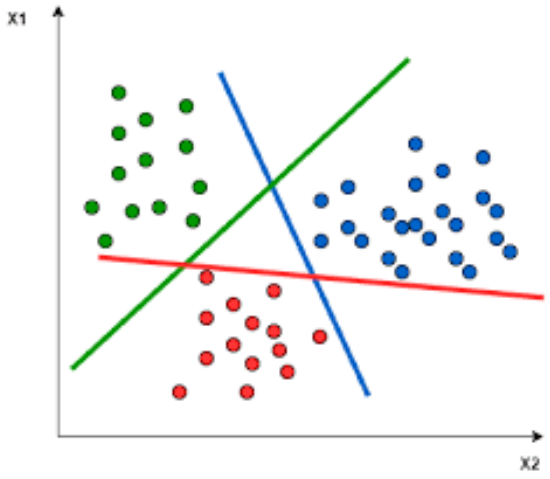


2. **One vs The Rest approach**

   This approach uses a binary SVM classifier to separate one class from all others. When finding the hyperplane that separates a single class from the rest, all points are taken into account, dividing them into two groups: one for the points of the target class and another for all other points (Figure 4.2). In this approach, if there are m different classes, this model will require training m different classifiers.

   Consequently, the complexity and computational time of the model grow linearly with the number of classes m. As in this case, each model must be trained with all the data. For the same number of classes, depending on the value of n, one method may be more computationally expensive than the other.

Figure 4.2
Graphical representation of the hyperplanes separating each class from the remaining ones in the One vs The Rest approach [9]



4.2.3. Kernel trick

As briefly mentioned before, the kernel trick allows SVM models to map the input data $\vec{x} \in \mathbb{R}^p$ into a higher-dimensional space $\mathbb{R}^N$ via a map $\phi : \mathbb{R}^p \to \mathbb{R}^N$. In the new space, known as "feature space", the data is more likely to be linearly separable. An important advantage of using the kernel trick is that is not necessary to know the expression of ϕ explicitly since according to (4.10), in order to classify a new data point $\vec{x_i}$, it is only necessary to know the scalar product among the rows of the dataset $\vec{x_j}\vec{x_k}$. Hence, it is merely required to know the expression of the scalar product $\phi(\vec{x_j}).\phi(\vec{x_k})$. The function $K(\vec{x}, \vec{y}) = \phi(\vec{x}).\phi(\vec{y})$ it is commonly known as kernel and is completely characterized by **Mercer's Theorem** [1]. According to this theorem, a function $K(\vec{x}, \vec{y})$ can be expressed as a scalar product $\phi(\vec{x}).\phi(\vec{y})$ if it satisfies **Mercer's Condition**. This condition states that the integral of $K(\vec{x}, \vec{y})$ multiplied by any pair of square-integrable functions $g(\vec{x}), g(\vec{y})$ over the entire input space must be non-negative.

$$\iint g(\vec{x})g(\vec{y})K(\vec{x}, \vec{y})d\vec{x}d\vec{y} \geq 0 \tag{4.11}$$

This condition is completely equivalent to these other two:

1. **Positive Definiteness:** For any set of coefficients c_i with $j = 1, ..., n$, it is satisfied that $\sum_{i,j} c_i c_j K(\vec{x_i}, \vec{x_j}) \geq 0$
2. **Symmetry:** It is satisfied $K(\vec{x}, \vec{y}) = K(\vec{y}, \vec{x})$, $\forall \vec{x}, \vec{y}$

In traditional Machine Learning, the most used kernels are:

1. **Linear kernel:** It is the most efficient and computationally fast kernel. It is suitable for high-dimensional datasets.

$$K(\vec{x}, \vec{y}) = \vec{x}\,\vec{y} \tag{4.12}$$

2. **Polynomial kernel:** It is effective for high-dimensional datasets with a relatively small number of observations. One of its main advantages is that it can capture non-linear relationships in the data. The computational cost increases with the degree of the polynomial α.

$$K(\vec{x}, \vec{y}) = (\vec{x}\,\vec{y} + c)^{\alpha} \tag{4.13}$$

3. **Gaussian Radial Basis Formula (RBF):** It is one of the most preferred and used kernel functions in SVM. It is usually chosen for highly non-linear data. It helps to make proper separations when there is no prior knowledge of data.

$$K(\vec{x}, \vec{y}) = e^{-\|\vec{x}-\vec{y}\|/2\sigma^2} \tag{4.14}$$

4. **Sigmoid kernel:** This kernel is also known as the Multilayer Perceptron kernel. An SVM model using this kernel is equivalent to a neural network with 2 layers. It is used for non-linear data, but it has the drawback that it is quite sensitive to the selection of the hyperparameters γ and c (4.15):

$$K(\vec{x}, \vec{y}) = tanh(\gamma \vec{x}\, \vec{y} + c) \tag{4.15}$$

4.3. Extrapolation of SVM to quantum

Quantum Support Vector Machine models represent a specialized form of SVM models that leverage quantum circuits to compute the kernel function $K(\vec{x}, \vec{y})$. In this context, a quantum kernel is completely characterized by a quantum circuit that maps the rows from the original data into a feature space of quantum states $|\psi_j\rangle$. This circuit is represented by a quantum gate $\Phi(\vec{x_j})$ parametrized by the rows of the dataset x_j and acts on the initial state of the quantum circuit. Usually, the initial state of the quantum circuit is $|\psi_j\rangle = |0\rangle^{\otimes n}$, that is, all the qubits are in the individual quantum state $|0\rangle$. Then, the expression of the resulting state after applying the quantum circuit is $|\widetilde{\psi_j}\rangle = \Phi(\vec{x_j})|\psi_j\rangle$.

In practice, computing the quantum kernel $K(\vec{x_i}, \vec{x_j})$ involves taking two rows of the dataset as vectors, mapping them into quantum states via $\Phi(\vec{x})$ and finally, compute the kernel function or "scalar product" among the resulting quantum states. To take advantage of the benefits offered by quantum computing, the kernel function employed must be easily obtainable with a quantum computer and must also satisfy Mercer's condition. Considering 2 quantum states $|x\rangle = \Phi(\vec{x})|0\rangle$ and $|y\rangle = \Phi(\vec{y})|0\rangle$, a possible quantum kernel is given by:

$$K(\vec{x}, \vec{y}) = |\langle x|y\rangle|^2 = |\langle 0|\Phi^\dagger(\vec{x})\Phi(\vec{y})0\rangle|^2 \tag{4.16}$$

The meaning of this equation can be interpreted as the probability (3.2) that all the qubits are in the individual quantum state $|0\rangle$ when measuring the quantum state $\Phi^\dagger(\vec{x})\Phi(\vec{y})0\rangle$. This quantum state is achieved by applying the quantum circuit $\Phi(\vec{x})$ to the initial state $|0\rangle^{\otimes n}$ followed by applying the quantum circuit $\Phi^\dagger(\vec{y})$ to the resulting state. Taking into account that in quantum computing, a circuit $\Phi(\vec{x})$ consists of multiple quantum gates, which are represented by unitary matrices (2.4), then, the circuit $\Phi^\dagger(\vec{x})$ have the same quantum gates as $\Phi(\vec{x})$ but they are applied from left to right instead.

It is important to check that the quantum kernel function defined in (4.16) fulfills **Mercer's condition**, that is, it is symmetric and positive definite. For that, we first prove that $\langle x|y\rangle$ is a kernel and then use the fact that the product of kernels is also a kernel [4]. As the inner product of quantum states is symmetric by definition, it is only left to prove that is positive definite:

$$\sum_{i,j} c_i c_j^* \langle x_i|x_j\rangle = \left(\sum_i c_i \langle x_i|\right)\left(\sum_j c_j^* |x_j\rangle\right) = \left\|\sum_i c_i |x_i\rangle\right\|^2 \geq 0. \tag{4.17}$$

The symbol $\|.\|$ in (4.17) denotes the norm of the vector associated with a quantum state, which, by definition, is always non-negative.

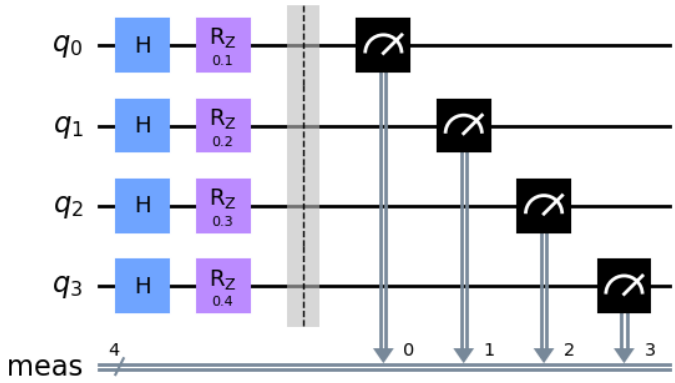
4.3.1. Quantum feature maps

In this section, we will introduce the most widely used types of quantum circuits $\Phi(\vec{x})$ employed for mapping the data into a space of quantum states:

1. **Angle encoding:** Angle encoding stands out as a foundational technique in quantum circuits, offering simplicity and effectiveness. When applied to a circuit with n qubits, angle encoding can accommodate n numerical inputs $x_1, x_2, ..., x_n$. This encoding method involves the application of rotation gates to each qubit, with the rotation angles determined by the respective x_j values. These x_j values essentially dictate the rotation angles, hence the name 'angle encoding'. In quantum circuits that use angle encoding, it can be used any rotational gate $R_X(\theta), R_Y(\theta), R_Z(\theta)$. However, when rotational gates $R_Z(\theta)$ are used, it is necessary to apply a Hadamard gate to each qubit before, since the action of a rotation gate $R_Z(\theta)$ on the zero quantum states has no effect at all. $R_Z(\theta)|0\rangle = |0\rangle$. One important aspect of using angle encoding is normalizing the variables x_j. Recalling that applying a quantum rotational gate $R_{X_i}(x_j)$ to a quantum state $|\psi\rangle$ can be physically interpreted as a rotation of x_j degrees around the X_i axe. Therefore, the values of the dataset's attributes should be within the closed interval $[0, 4\pi]$. Nevertheless, normalizing the variables within this interval has one drawback, which is that the points with values 0 and 2π are mapped to the same quantum state since $R_{X_i}(0) = R_{X_i}$. This could lead to data points that are very different being indistinguishable. To solve this, the variables can be normalized within the smallest interval, such as $[0, 1]$.

2. **Amplitude encoding:** The Amplitude Encoding map is far more complicated and less studied than Angle Encoding and is suitable for high-dimensional datasets since it can map 2^n points into a quantum circuit with only n qubits. In other words, if the dataset has n predictors, the Amplitude Encoding map will use $\lceil log_2(n) \rceil$ qubits. This map transforms each row of the dataset into the following quantum state:

Example of angle encoding quantum circuit using Hadamard and R_Z quantum gates



$$|\Phi(\vec{x})\rangle = \frac{1}{\sqrt{\sum_l x_l^2}} \sum_{l=0}^{n-1} x_l |l\rangle, \tag{4.18}$$

where the factor $\frac{1}{\sqrt{\sum_l x_l^2}}$ ensures the resulting quantum state is normalized. The quantum circuit that implements this map is very complex and difficult to describe in terms of elementary quantum gates [5]. Furthermore, it goes beyond the scope of this project and requires deeper knowledge of quantum computing.

3. **ZZ Circuit:** As with the Angle Encoding map, the ZZ map takes n variables and transforms them into the feature space using a quantum circuit with n qubits. The ZZ feature map derives its name from the interaction term involving Pauli-Z matrices. One of its primary advantages is the creation of entanglement between qubits corresponding to different attributes. This entanglement enables the model to capture complex patterns and relationships in the data, facilitating the learning of highly non-linear boundaries and often outperforming classical SVMs in such cases.

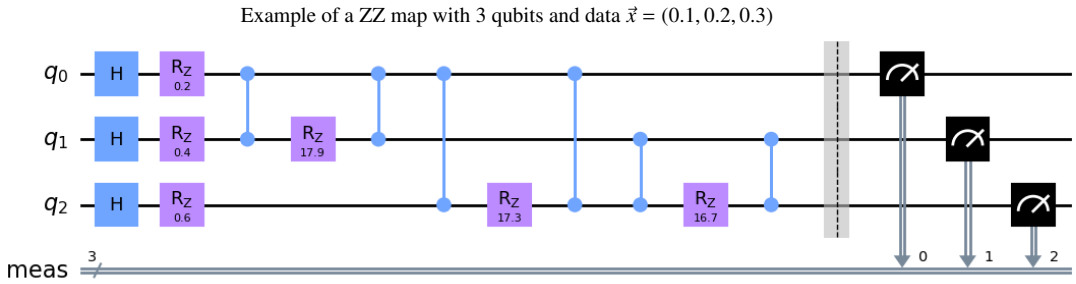
The construction of the quantum circuit implemented by the ZZ map proceeds through the following steps:

1. Begin by applying a Hadamard gate on each qubit.
2. Next, apply a rotational gate $R_Z(2x_j)$ j-th qubit.
3. For each pair of qubits $\{i, j\}$, apply the following quantum gates:

   - Apply a CNOT gate with qubit i as the control and qubit j as the target.

   -Apply a rotational gate $R_Z(2(\pi - x_i)(\pi - x_j))$ on qubit j

   - Apply another controlled CNOT gate.



As with models employing Angle Encoding, ZZ maps contain several rotational gates, and therefore, variable normalization plays an important role. Typically, in the literature, variables are normalized to the intervals $[0, 1]$ and $[0, 3]$ [1].

During the testing of these models with synthetic datasets and the comparison of performance among different models, we will only use these maps for QSVMs. Nevertheless, there are infinite possibilities when creating a quantum kernel, and the diverse selection of feature maps for QSVMs is a current area of research in the field of quantum machine learning.

In [6], additional examples of quantum maps are explored, along with proposals for comparing different quantum kernels for quantum machine learning purposes.

5. CONTRIBUTIONS TO THE QSVM: METHODOLOGY AND CASE STUDY DESIGN

In this section, the Quantum Support Vector Machine (QSVM) will be applied, using the quantum mappings and encodings discussed earlier, to several datasets with distinct properties for both binary and multiclass classification. Alongside the QSVM, classical machine learning algorithms such as Logistic Regression, Random Forest, and Support Vector Machines (SVM) with linear, polynomial, RBF, and sigmoid kernels will also be applied. The performance of the QSVM models will be compared with these classical algorithms, aiming to identify datasets where QSVM outperforms them, determine the underlying patterns and characteristics of these datasets, and identify which quantum map or encoding is most suitable for each situation.

More specifically, for each dataset, QSVMs will be evaluated using distinct maps: Amplitude Encoding, Angle Encoding with normalization intervals of $[0, 1]$ and $[0, 4\pi]$, and ZZ maps with normalization intervals of $[0, 1]$ and $[0, 3]$. The idea behind testing different intervals for Angle Encoding and ZZ maps is that, as mentioned in the theory, the performance of these models can be sensitive to the range in which the predictor variables are normalized, and it would be beneficial to empirically verify this fact.

To test and compare the performance of the models across datasets with varied characteristics, such as the number of variables and class proportions, three different metrics will be employed: accuracy, F_1 score, and the Matthews correlation coefficient. Additionally, the execution time required for model training will be measured, since as it will be discussed later, one of the primary drawbacks of Quantum Machine Learning models nowadays is their extended execution time.

Accuracy: The accuracy metric measures the overall correctness of predictions made by a model. It computes the proportion of correctly classified instances out of the total number of observations in the dataset.

In binary classification:

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{5.1}$$

In multiclass classification:

$$accuracy = \frac{\sum_i C_{ii}}{\sum_i \sum_j C_{ij}}, \tag{5.2}$$

where the coefficients C_{ij} are the elements of the confusion matrix.

F1 score: This metric is defined as the harmonic mean between precision, which is the ratio of true positive predictions to the total number of positive predictions, and recall, which is the ratio between of predicted positives and the actual number of positives. The possible values of F_1 score range from 0 to 1. This metric is often used for comparing the performance of different models in binary and multiclass classification tasks, even when the classes are not balanced.

$$F_1 = \frac{2precision * recall}{precision + recall} \tag{5.3}$$

In multilabel classification, the F_1 score can be obtained following two different approaches: micro-averaging and macro-averaging.

1. **Micro-averaging:** In micro-averaging, the F_1 score is computed by taking into account the contributions of all classes collectively. It essentially computes a global average by adding the individual true positives, false positives, and false negatives across all classes before calculating precision and recall.

2. **Macro-averaging:** In macro-averaging, the F_1 score is calculated for each class independently, and then the average of these scores is taken. This treats each class equally, regardless of its frequency.

Mathews correlation:

The Mathews Correlation Coefficient (MCC) is a powerful metric for evaluating the performance of models in binary problems taking into account the four elements of the confusion matrix (true positives, false positives, true negatives, and false negatives) providing a more representative picture of classifier performance compared to other metrics such as accuracy and F_1 score, which ignores true negatives and accuracy. Mathematically, the MCC is computed as follows:

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)}} \tag{5.4}$$

According to [7], the Matthews correlation coefficient provides more accurate and informative results than accuracy and the F_1 score in binary classification and it is a more reliable metric since MCC is high only if the prediction obtained good results in all of the four confusion matrix categories.

In a multi-class classification problem with m classes, the expression for the MCC [8] is generalized as:

$$\text{MCC} = \frac{\sum_k \sum_l \sum_m C_{kk} C_{lm} - C_{kl} C_{mk}}{\sqrt{\left(\sum_k t_k p_k\right) - \left(\sum_k t_k^2\right)} \sqrt{\left(\sum_k p_k n\right) - \left(\sum_k p_k^2\right)}}, \tag{5.5}$$

where:

- C_{ij} is the matrix element in the i-th row and the j-th column of the confusion matrix, representing the number of samples known to be in class i and predicted to be in class j.

- t_k is the total number of true instances of class k, that is, the sum of the entries in the k-th row of the confusion matrix.

- p_k denotes the total number of predicted instances of class k, which can be computed as the sum of the k-th column of the confusion matrix.

Finally, n stands for the total number of predicted samples, which can be computed as the sum of all the elements in the confusion matrix C.

5.1. Implementation of QSVM models in Python

Although the full potential of quantum computers is not yet realized, ideal quantum computers can be simulated using classical computers, at least for a small number of qubits [1].

We will briefly discuss two of the most widely used quantum frameworks for implementing and simulating quantum circuits, which will be used in this project:

1. **Qiskit**

   Qiskit is a quantum software framework developed by IBM. It uses Python as the host language and has extensions for well-known machine learning libraries such as PyTorch. One of the main features of this framework is that it provides a comprehensive set of tools for implementing quantum circuits, simulating them, and even running them on real IBM quantum computers using an IBM account.

   In this project, the Qiskit library will be used to implement a ZZ map and compare the performance of this quantum encoding with the ZZ maps defined in Pennylane. The quantum circuit representing this encoding is simulated using the Qiskit Aer simulator. The Qiskit Aer simulator is a high-performance simulation framework within the Qiskit ecosystem, designed to simulate quantum circuits. It allows users to run quantum experiments on their classical computers with high fidelity and efficiency. In Qiskit, the QSVM model with a ZZ map can be implemented easily without manually defining the kernel. Details regarding the implementation and the code of the ZZ map in Qiskit can be found in [Appendix I].

2. **Pennylane**

   `PennyLane` is a quantum framework developed by Xanadu, which also uses Python as the host language. This software is specifically designed for quantum machine learning. One of its greatest advantages is its compatibility with many Python libraries, such as `scikit-learn`, `Keras`, `TensorFlow`, and `PyTorch`.

   Additionally, `Pennylane` provides a great deal of built-in maps feature maps, including Amplitude encoding and Angle encoding ones. Besides, it allows creating your own custom feature map. For these reasons, the QSVM models and the maps with the quantum circuits Amplitude Encoding, Angle Encoding, and ZZ map with different ranges of normalization of the predictor variables will be implemented with `PennyLane`, which also allows using these kernels in the SVM models of `scikit-learn`. The only QSVM model that will also be trained in `Qiskit` is the ZZ to observe the differences with the corresponding model in the `PennyLane` library. The details of the creation of the quantum circuits for each feature map and their implementation in `scikit-learn` can be found in the Appendix.

So far, we have discussed the different quantum and classical machine learning models to be employed, as well as the various metrics used to evaluate the performance of each model. It has also been specified which quantum computing libraries were used to define the different quantum kernels. Finally, we will discuss the procedure for evaluating the performance of each method, as well as some technical limitations in the case of datasets with very high dimensions.

As stated earlier, the full potential and the main advantages of Quantum Computing are not available yet and simulating quantum circuits takes a significant amount of time. For this reason, hyperparameter tuning will not be performed on the trained models as this would significantly increase execution times. Instead, the various QSVM models will be trained with the default hyperparameter C that `scikit-learn` uses ($C = 1$). For evaluating the performance of the models, it will be carried out a $k-$ fold cross-validation using $k = 5$, as a higher number of folds would drastically increase the computational time.

Finally, it is worth noting that both in `Pennylane` and `Qiskit`, with current simulators, quantum circuits of approximately 13 qubits can be implemented. This implies that Angle encoding and ZZ maps can only be used to train models with datasets containing a maximum of 13 variables, which is quite limited. In contrast, Amplitude encoding can handle datasets with up to $2^{13} = 8192$ variables, which is significantly higher and allows for studying cases with high-dimensional data. For high-dimensional datasets, the plan is to apply PCA (Principal Component Analysis) to the predictors, iterating over the number of components from $n_{component} = 2$ to $n_{component} = 13$. For each number of components, metrics such as execution time and the proportion of variability explained will be calculated, with the best-performing number of components shown in the table along with these results.

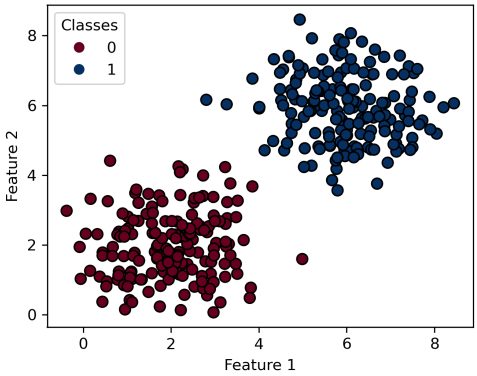
6. RESULTS

Most datasets used to test the performance of support vector machines have been generated synthetically, as synthetic data provides complete control over data characteristics. The remaining datasets are sourced from [10].

6.1. Binary classification

In this dataset, predictors are generated from multivariate normal distributions; Class 1 has mean vector $\vec{\mu}_1 = (2, 2)$ and Class 2 has $\vec{\mu}_2 = (6, 6)$, with covariance matrices having diagonal elements of 1, ensuring uncorrelated predictors and perfect linear separability (see Figure 6.1). The dataset includes $n = 336$ observations with $p = 2$ predictors.

Figure 6.1
Graphical Representation of Linearly Separable Dataset



Method	Accuracy	F1 Score	Matthews Correlation	Time(s)
QSVM Amplitude Encoding	0.588	0.695	0.344	225.567
QSVM Angle Encoding [0,1]	1.000	1.000	1.000	147.724
QSVM Angle Encoding [0,4π]	1.000	1.000	1.000	118.339
QSVM ZZ [0,1]	1.000	1.000	1.000	193.220
QSVM ZZ [0,3]	1.000	1.000	1.000	201.360
QSVM Qiskit	1.000	1.000	1.000	220.647
SVM Linear	1.000	1.000	1.000	0.000
SVM Poly	1.000	1.000	1.000	0.000
SVM RBF	1.000	1.000	1.000	0.000
SVM Sigmoid	0.004	0.08	-0.91	0.000
Logistic Regression	1.000	1.000	1.000	0.000
Random Forest	1.000	1.000	1.000	0.000

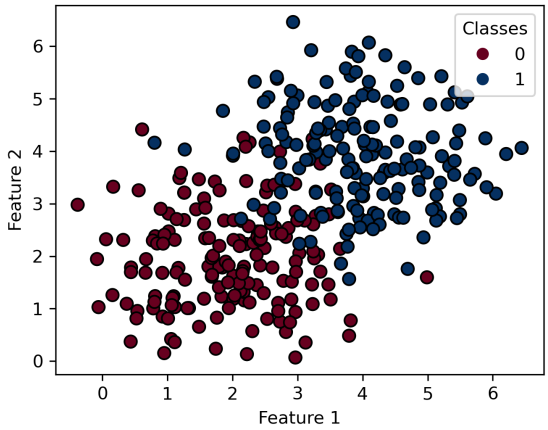
Table 6.1
Results of QSVM on a linearly separable dataset

As shown in Table 6.1, nearly all models, both classical and quantum, achieve 100% accuracy in predicting both classes. The only exceptions are the QSVM with Amplitude Encoding, which performs slightly better than a random assignment of classes, and the SVM with a sigmoid kernel, which achieves an accuracy close to 0%, indicating very poor performance.

Now, a dataset similar to the previous one is being tested, but with observations corresponding to different classes now closer together; specifically, the vector means are (2,2) and (4,4). Consequently, observations belonging to different classes overlap, resulting in data that is not perfectly linearly separable (see Figure 6.2).

Figure 6.2

Graphical Representation of Linearly Separable Dataset with overlapping between classes



Method	**Accuracy**	**F1 Score**	**Matthews Correlation**	**Time(s)**
QSVM Amplitude Encoding	0.558	0.681	0.293	155.367
QSVM Angle Encoding [0,1]	0.912	0.906	0.823	78.912
QSVM Angle Encoding [0,4π]	0.647	0.625	0.292	86.036
QSVM ZZ [0,1]	0.897	0.888	0.793	193.553
QSVM ZZ [0,3]	0.897	0.888	0.793	169.802
QSVM Qiskit	0.897	0.892	0.794	210.311
SVM Linear	0.900	0.890	0.790	0.000
SVM Poly	0.880	0.870	0.760	0.000
SVM RBF	0.910	0.910	0.820	0.000
SVM Sigmoid	0.100	0.120	-0.790	0.000
Logistic Regression	0.897	0.888	0.793	
Random Forest	0.912	0.906	0.823	1.628

Table 6.2

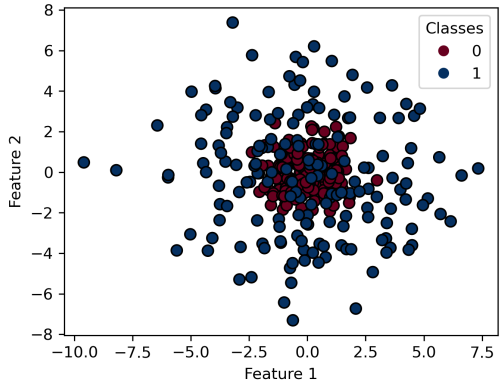
Results of QSVM on a linearly separable dataset with overlapping between classes

In this dataset, the results show some notable differences (see Table 6.2). None of the algorithms are capable of achieving 100% accuracy anymore; RandomForest and QSVM with Angle encoding and a normalization range of $[0, 1]$ exhibit the best metrics. As in the previous case, the algorithms performing worst are QSVM with Amplitude Encoding and SVM with sigmoid kernel. An important observation is that, for this specific dataset, the normalization range of variables appears to have an influence, particularly in QSVM with Angle Encoding.

The following dataset has $p = 2$ predictors and $n = 336$ observations. The relationship between the predictors and the response is non-linear. Covariates of different classes are normally distributed with the same mean but different variances. The inner circle (class 0) has a mean vector $\vec{\mu} = (0, 0)$ and a diagonal covariance matrix with ones on the diagonal. The outer circle (class 1) has the same mean but a different diagonal covariance matrix with elements equal to 9. This causes significant class overlap, leading to non-linear separability (see Figure 6.3).

Figure 6.3

Graphical Representation of a non-linearly separable dataset with concentric Gaussian distributions



Method	Accuracy	F1 Score	Matthews Correlation	Time(s)
QSVM Amplitude Encoding	0.662	0.439	0.414	164.475
QSVM Angle Encoding [0,1]	0.471	0.639	0.000	147.724
QSVM Angle Encoding [0,4π]	0.838	0.807	0.686	118.339
QSVM ZZ [0,1]	0.810	0.754	0.645	174.97
QSVM ZZ [0,3]	0.823	0.778	0.670	194.167
QSVM Qiskit	0.852	0.815	0.734	174.34
SVM Linear	0.460	0.630	-0.130	0.000
SVM Poly	0.840	0.810	0.690	0.000
SVM RBF	0.910	0.900	0.830	0.000
SVM Sigmoid	0.470	0.640	0.00	0.000
Logistic Regression	0.440	0.472	-0.109	0.000
Random Forest	0.867	0.852	0.736	8.73

Table 6.3

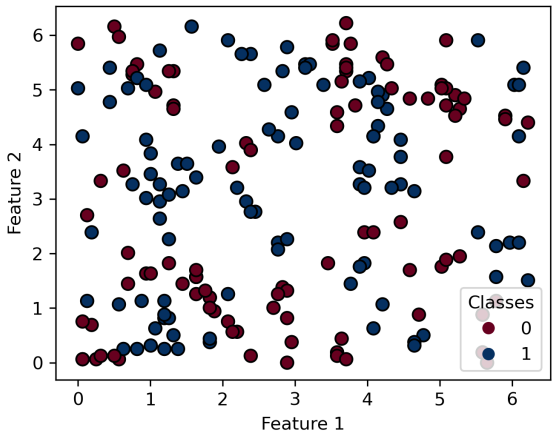
Results of QSVM on a non-linear separable dataset with concentric Gaussian distributions

As shown in Table 6.3, the algorithms with the best performance are Random Forest and SVM with RBF kernel, surpassing the quantum models using various feature maps. QSVM achieves significantly better results compared to Logistic Regression and SVM with a linear kernel, both of which perform poorly due to the dataset's highly non-linear nature. Furthermore, the metrics for QSVM with Angle Encoding show notable differences for both normalization ranges.

Following the trend of testing highly non-linear datasets, this time, a dataset from the qiskit library is employed. Labels are assigned using a quantum circuit, and the dataset is perfectly balanced. It consists of $n = 200$ observations and $p = 2$ predictors. The details of the quantum circuit and the rule implemented to assign the labels can be found in https://docs.quantum.ibm.com/api/qiskit/0.19/qiskit.ml.datasets.ad_hoc_data

Figure 6.4

Graphical representation of the dataset with classes generated using quantum circuits.



Method	Accuracy	F1 Score	Matthews Correlation	Time(s)
QSVM Amplitude Encoding	0.450	0.313	-0.110	58.46
QSVM Angle Encoding [0,1]	0.400	0.400	-0.2	33.062
QSVM Angle Encoding [0,4π]	0.775	0.791	0.548	31.35
QSVM ZZ [0,1]	0.550	0.625	0.110	66.259
QSVM ZZ [0,3]	0.525	0.558	0.045	69.877
QSVM Qiskit	0.625	0.651	0.253	69.877
SVM Linear	0.450	0.420	-0.110	0.000
SVM Poly	0.600	0.670	0.22	0.000
SVM RBF	0.570	0.590	0.150	0.000
SVM Sigmoid	0.470	0.490	-0.05	0.000
Logistic Regression	0.275	0.216	-0.45	0.000
Random Forest	0.75	0.762	0.502	14.438

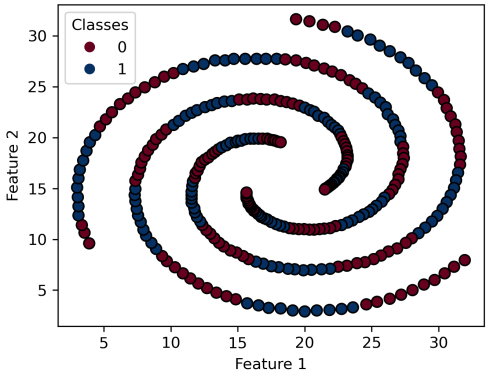
Table 6.4

Results on the quantum dataset

Based on Figure 6.4, there appears to be no discernible pattern, and it seems as if the classes were assigned almost randomly. Most machine learning algorithms yield poor metrics, with the exceptions being Random Forest, Angle encoding with normalization in the range $[0, 4\pi]$, and the ZZ map from Qiskit. Furthermore, in this specific case, the QSVM with Angle encoding is the best-performing model, outperforming all classical SVM models and Random Forest, despite their similar performance 6.4.

Next, a highly non-linear dataset from [10] has been chosen, which is part of a repository containing several datasets commonly used as benchmarks for clustering and machine learning algorithms. Specifically, this dataset contains 2 predictors and $n = 312$ observations. When both predictors are plotted, they form a spiral, with the classes alternating along different segments of the spiral. More specifically, the spiral formed by both predictors is divided into 30 segments, and the classes alternate with each segment of the spiral.

Figure 6.5
Graphical representation of the spiral dataset.



Method	Accuracy	F1 Score	Matthews Correlation	Time(s)
QSVM Amplitude Encoding	0.539	0.171	-0.066	185.555
QSVM Angle Encoding [0,1]	0.619	0.143	0.216	84.78
QSVM Angle Encoding [0,4π]	0.635	0.530	0.23	78.31
QSVM ZZ [0,1]	0.539	0.383	0.023	174.49
QSVM ZZ [0,3]	0.667	0.571	0.302	174.64
QSVM Qiskit	0.603	0.561	0.207	180.150
SVM Linear	0.540	0.450	0.060	0.000
SVM Poly	0.480	0.560	0.060	0.000
SVM RBF	0.590	0.280	0.080	0.000
SVM Sigmoid	0.400	0.300	-0.230	0.000
Logistic Regression	0.539	0.453	0.056	0.000
Random Forest	0.889	0.857	0.7706	19.672

Table 6.5
Results of the spiral dataset

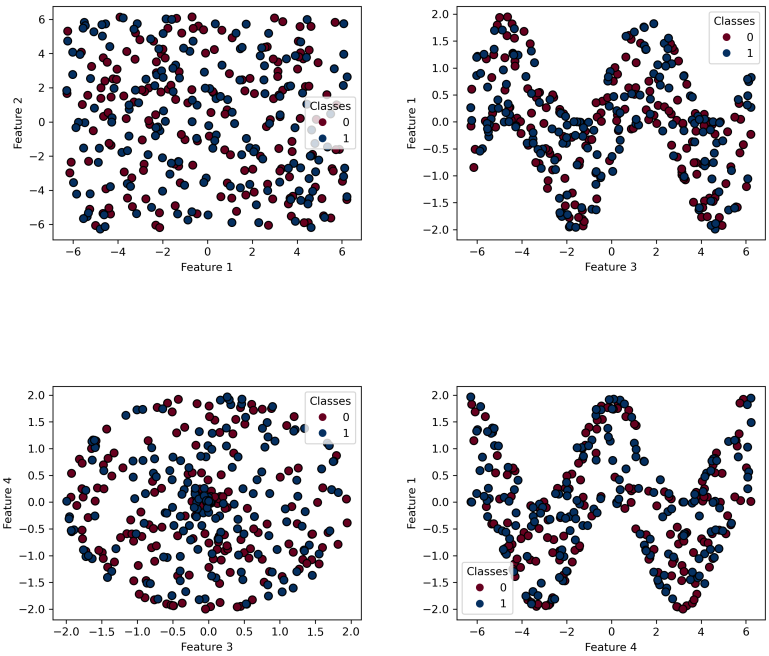
From Table 6.5 , it is seen that Random Forest stands out as the most effective classifier in this dataset, showing superior performance across all metrics. Following Random Forest, some QSVM models also performed well, notably the Angle Encoding with a normalization range of $[0, 4\pi]$ and the ZZ map with a normalization range of $[0, 3]$, outperforming Logistic Regression and classical SVM models.

So far, datasets with two predictors and a highly non-linear relationship between the predictors and the target have been studied. Next, a couple of datasets with higher dimensionality will be examined.

This one contains $n = 336$ observations and $p = 4$ predictors with a high nonlinear relationship between the variables and the response. The first two predictors are sampled from a uniform distribution $U(-2\pi, 2\pi)$. The third and fourth predictors are then generated by applying the transformation $sin(X_1) \pm Cos(X_2)$ for one class and $sin(X_1) \mp Cos(X_2)$ for the other class (see Figure 6.6).

Figure 6.6

Plots between some of the predictors of the dataset.



Method	Accuracy	F1 Score	Matthews Correlation	Time(s)
QSVM Amplitude Encoding	0.529	0.652	0.161	188.625
QSVM Angle Encoding [0,1]	0.515	0.602	0.068	93.875
QSVM Angle Encoding [0,4π]	0.721	0.716	0.444	97.000
QSVM ZZ [0,1]	0.588	0.533	0.169	602.0625
QSVM ZZ [0,3]	0.500	0.433	-0.0105	546.156
QSVM Qiskit	0.647	0.684	0.326	546.156
SVM Linear	0.560	0.590	0.140	0.000
SVM Poly	0.570	0.650	0.200	0.000
SVM RBF	0.620	0.691	0.300	0.000
SVM Sigmoid	0.560	0.660	0.200	0.000
Logistic Regression	0.529	0.579	0.079	0.000
Random Forest	0.867	0.897	0.745	2.110

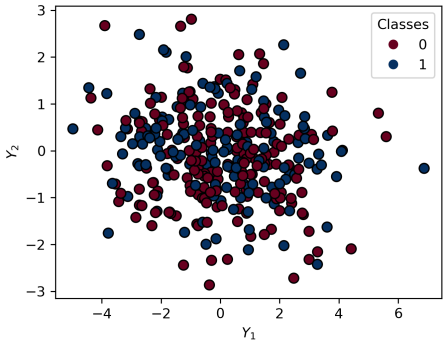
Table 6.6
Results of the dataset that contains 4 *predictors*

As in the previous dataset (see Table 6.5), the Random Forest algorithm performs best, with metrics significantly superior to the others. Following this model, the QSVMs with Angle Encoding in the range $(0, 4\pi)$ and the ZZ map from Qiskit have better metrics than the Logistic Regression model and the classical QSVMs (see Table 6.6).

The final synthetic dataset that will be analyzed for binary classification contains 100 predictors and $n = 336$ observations. The values of the predictors are sampled from a multivariate normal distribution with significant correlation among them. Specifically, the correlation between a pair of predictors X_i and X_j is given by $\rho^{|i-j|}$, where the value of ρ is set to 0.9.

For generating the response variable, an exponential function is first applied to each predictor. This results in transformed predictor values. Subsequently, a linear combination $X\beta$ is formed using these transformed predictors. Finally, a logit transformation is applied to convert this linear combination into a probability. The label is then assigned based on whether this probability is greater than or less than $p = 0.5$. A plot of the first two principal components can be seen in Figure 6.7

Figure 6.7
Plot of two first principal components.



Method	Accuracy	F1 Score	Matthews Correlation	Time(s)
QSVM Amplitude Encoding	0.558	0.0625	0.133	2261.484
QSVM Angle Encoding [0,1]	0.632	0.444	0.26 4	254.22
QSVM Angle Encoding [0,4π]	0.573	0.408	0.120	119.437
QSVM ZZ [0,1]	0.617	0.518	0.219	857.765
QSVM ZZ [0,3]	0.588	0.461	0.155	1171.025
QSVM Qiskit	0.573	0.171	0.148	6434.26
SVM Linear	0.720	0.650	0.440	0.000
SVM Poly	0.690	0.630	0.370	0.000
SVM RBF	0.680	0.520	0.370	0.000
SVM Sigmoid	0.540	0.000	0.000	0.000
Logistic Regression	0.662	0.596	0.312	0.004
Random Forest	0.662	0.488	0.337	45.36

Table 6.7
Performance results on the dataset that contains 100 *predictor variables*

Method	$n_{\text{component}}$	Explained Variance Ratio
QSVM Amplitude Encoding	-	-
QSVM Angle Encoding [0,1]	5	0.31
QSVM Angle Encoding [0,4π]	7	0.31
QSVM ZZ [0,1]	6	0.25
QSVM ZZ [0,3]	7	0.25
QSVM Qiskit	9	0.34

Table 6.8
Optimal number of principal components and the corresponding explained variance ratio for each algorithm

As shown in 6.7, all QSVM models are outperformed by classical algorithms such as RandomForest, Logistic Regression, and SVM with polynomial and linear kernels, which demonstrate superior metrics. From 6.8, it is also noteworthy that across all machine learning algorithms, the best performance is achieved with a minimal number of principal components relative to the dataset's dimensionality, and the percentage of variability explained by this number of components is not very high.

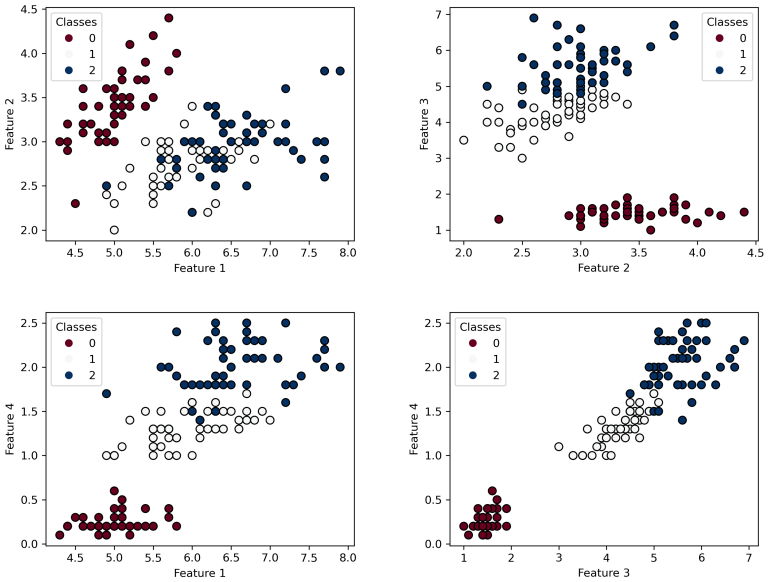
6.2. Multiclass classification

In this section, the **One vs One** and **One vs The Rest** approaches explained in Section 4.2.2 will be used to perform multiclass classification using Quantum and Classical Support Vector Machine models. Consequently, two tables will be presented for each dataset, each showcasing the respective metrics for these methods

The first multiclass dataset to be analyzed is the Iris dataset, well-known in the fields of statistics and machine learning, and often used as a benchmark for evaluating new machine learning methods. This dataset was initially introduced by Ronald Fisher in [Insert reference here] as an example of discriminant analysis. It comprises 150 specimens of iris flowers categorized into three species: Setosa, Versicolor, and Virginica. The dataset includes four features corresponding to different attributes of the iris flowers: sepal length (*cm*), sepal width (*cm*), petal length (*cm*), and petal width (*cm*). Additionally, all three species are equally represented in the dataset..

Figure 6.8

Graphical representation of the relation between some of the features of the Iris dataset.



One vs One approach

Method	Accuracy	F1 Score	Matthews Correlation	Time(s)
QSVM Amplitude Encoding	0.567	0.517	0.555	52.984
QSVM Angle Encoding [0,1]	1.000	1.000	1.000	26.031
QSVM Angle Encoding [0,4π]	0.833	0.838	0.760	26.453
QSVM ZZ [0,1]	1.000	1.000	1.000	139.781
QSVM ZZ [0,3]	0.967	0.967	0.951	152.406
QSVM Qiskit	1.000	1.000	1.000	145.172
SVM Linear	1.000	1.000	1.000	0.000
SVM Poly	1.000	1.000	1.000	0.000
SVM RBF	1.000	1.000	1.000	0.000
SVM Sigmoid	0.333	0.333	0.291	0.000
Logistic Regression	1.000	1.000	1.000	0.000
Random Forest	1.000	1.000	1.000	14.297

Table 6.9

Results on Iris dataset using One vs One approach

One vs Rest approach

Method	Accuracy	F1 Score	Matthews Correlation	Time(s)
QSVM Amplitude Encoding	0.567	0.517	0.555	113.078
QSVM Angle Encoding [0,1]	0.800	0.809	0.752	56.859
QSVM Angle Encoding [0,4π]	0.733	0.742	0.760	64.734
QSVM ZZ [0,1]	0.967	0.960	0.950	365.525
QSVM ZZ [0,3]	0.967	0.960	0.950	368.525
QSVM Qiskit	1.000	1.000	1.000	349.890
SVM Linear	0.833	0.841	0.787	0.000
SVM Poly	1.000	1.000	1.000	0.000
SVM RBF	1.000	1.000	1.000	0.000
SVM Sigmoid	0.567	0.464	0.339	0.000
Logistic Regression	1.000	1.000	1.000	0.000
Random Forest	1.000	1.000	1.000	14.297

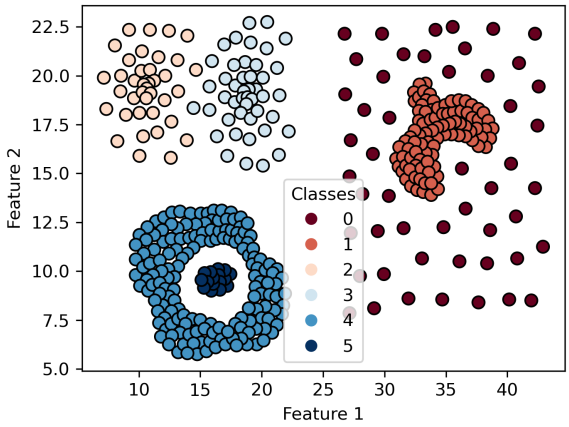
Table 6.10
Results on Iris dataset using One vs Rest approach

As it is shown in Table 6.9, using the One vs One approach method, most algorithms achieve 100% in all metrics except for QSVM with Amplitude encoding, Angle Encoding in the range $[0, 4\pi]$, ZZ map in the range $[0, 3]$, and SVM with a sigmoid kernel, which exhibits very poor performance. On the other hand, using the One vs The Rest approach (see Table 6.10), there is a decrease in the metrics of most QSVM models and in the SVM with the linear kernel; moreover, the execution times are longer. Both the reduction in predictive power of the models and the increase in execution time when using the One vs Rest approach, compared to the One vs One approach, are consistently observed across all datasets studied. In the Iris dataset, observations of flowers belonging to different classes appear to be linearly separable [Reference to corresponding plot]. Next, datasets will be analyzed following a similar approach as used for binary classification, focusing on cases where the relationship between predictor variables and target classes is highly non-linear.

The first non-linear dataset analyzed is sourced from [10], commonly known in the literature as the Compound's dataset. It serves as a benchmark for clustering and classification algorithms, comprising $n = 400$ observations and $p = 2$ predictor variables across 6 distinct classes. While some pairs of classes may be linearly separable, others exhibit highly nonlinear relationships (see Figure 6.9).

Figure 6.9

Graphical representation of Compound's dataset



One vs One approach

Method	Accuracy	F1 Score	Matthews Correlation	Time(s)
QSVM Amplitude Encoding	0.388	0.201	0.101	493.500
QSVM Angle Encoding [0,1]	0.825	0.589	0.768	273.703
QSVM Angle Encoding [0,4π]	0.613	0.391	0.453	273.375
QSVM ZZ [0,1]	0.900	0.755	0.866	580.720
QSVM ZZ [0,3]	0.825	0.554	0.757	607.234
QSVM Qiskit	0.862	0.706	0.814	1097.438
SVM Linear	0.838	0.631	0.786	0.000
SVM Poly	0.900	0.746	0.866	0.000
SVM RBF	0.975	0.975	0.966	0.000
SVM Sigmoid	0.200	0.055	-0.357	0.000
Logistic Regression	0.838	0.674	0.781	0.001
Random Forest	0.988	0.988	0.983	17.000

Table 6.11

Results on Compound's dataset using One vs One approach

Among the QSVMs employed, those using a ZZ map achieve the best metrics, although they are outperformed by the Random Forest model and SVMs with polynomial and RBF kernels. Notably, the SVM with a sigmoid kernel has quite low metrics, performing worse than selecting a class at random (see Table 6.11).

One vs Rest approach

Method	Accuracy	F1 Score	Matthews Correlation	Time(s)
QSVM Amplitude Encoding	0.338	0.319	0.092	1580.720
QSVM Angle Encoding [0,1]	0.712	0.353	0.610	825.390
QSVM Angle Encoding [0,4π]	0.562	0.296	0.379	809.343
QSVM ZZ [0,1]	0.875	0.704	0.833	1813.890
QSVM ZZ [0,3]	0.838	0.551	0.778	1777.937
QSVM Qiskit	0.850	0.689	0.800	3897.843
SVM Linear	0.712	0.353	0.610	0.000
SVM Poly	0.850	0.663	0.797	0.000
SVM RBF	0.912	0.771	0.882	0.000
SVM Sigmoid	0.000	0.000	-0.403	0.000
Logistic Regression	0.838	0.674	0.781	0.001
Random Forest	0.988	0.988	0.983	17.000

Table 6.12
Results on Compound's dataset using One vs Rest approach

With the One vs Rest approach, the same pattern is observed: the best models are the Random Forest and SVMs with polynomial kernels. However, overall, all SVM models perform worse than in the One-vs-One approach, except for the ZZ Map from Qiskit (see Table 6.12).

As in the case of binary classification, several Quantum Support Vector machine models performed well on the dataset where there was overlap between the two classes, making it more difficult to distinguish them. In this case, a synthetic dataset with three classes is generated, where the predictors and the response variable have a simple relationship. Still, the classes overlap, making them harder to distinguish. The noise is added to the dataset as white noise at the boundary between classes.

To ensure efficient execution times, the generated dataset comprises $n = 200$ observations. To enhance the dataset's complexity compared to previous iterations, the number of predictors has been increased from $p = 2$ to $p = 8$. Each predictor's values are generated from a normal distribution with a mean $\mu = 0$ and standard deviation $\sigma = 10$. The class labels are assigned based on the following criteria:

The sum s_1 of the first 4 predictors includes added normal noise with a standard deviation of 3. Similarly, the sum s_2 of the last 4 predictors incorporates normal noise with a standard deviation of 3.

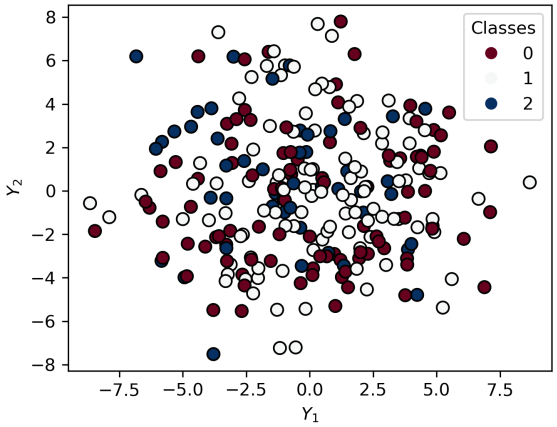
- Class 1 is assigned if $s_1 > 20$ and $s_2 > 15$

- Class 2 is assigned if $s_2 > 15$ and $s_1 \leq 20$

- Class 3 is assigned for the rest of values of s_1 and s_2.

Figure 6.10

Graphical representation of the first two principal components of the dataset with added white noise (σ = 3)



One vs One approach

Method	Accuracy	F1 Score	Matthews Correlation	Time(s)
QSVM Amplitude Encoding	0.660	0.655	0.459	249.281
QSVM Angle Encoding [0,1]	0.720	0.726	0.551	119.531
QSVM Angle Encoding [0,4π]	0.360	0.313	-0.080	126.531
QSVM ZZ [0,1]	0.480	0.363	0.152	2084.937
QSVM ZZ [0,3]	0.360	0.312	-0.058	1975.797
QSVM Qiskit	0.420	0.360	0.057	2090.547
SVM Linear	0.700	0.711	0.553	0.000
SVM Poly	0.720	0.740	0.556	0.000
SVM RBF	0.720	0.749	0.556	0.000
SVM Sigmoid	0.420	0.276	-0.012	0.000
Logistic Regression	0.660	0.672	0.516	0.000
Random Forest	0.680	0.717	0.482	18.641

Table 6.13

Results on the dataset with added white noise (σ = 3) using One vs One approach

The models with the highest metrics include the QSVM with Angle Encoding in the range $[0, 1]$, as well as the linear, polynomial, and RBF SVMs. Despite the QSVM with Angle Encoding achieving higher accuracy than the linear SVM and equivalent accuracy to the polynomial and RBF SVMs, its Matthews correlation coefficient is lower. This coefficient proves more suitable for model comparison in this scenario, given that the classes are not perfectly balanced.

One vs Rest approach

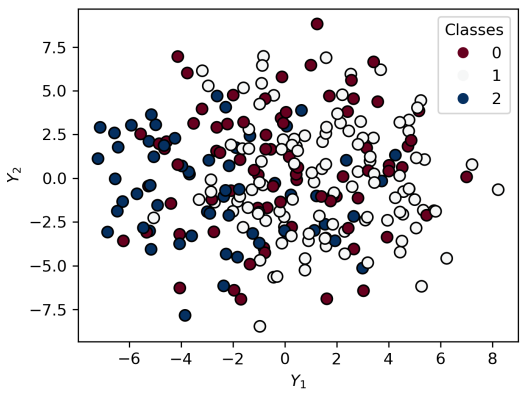
Method	Accuracy	F1 Score	Matthews Correlation	Time(s)
QSVM Amplitude Encoding	0.600	0.584	0.368	569.797
QSVM Angle Encoding [0,1]	0.700	0.712	0.523	280.094
QSVM Angle Encoding [0,4π]	0.340	0.295	-0.093	276.578
QSVM ZZ [0,1]	0.340	0.255	-0.049	4222.412
QSVM ZZ [0,3]	0.360	0.339	-0.034	4144.953
QSVM Qiskit	0.400	0.345	0.017	4363.890
SVM Linear	0.640	0.616	0.437	0.000
SVM Poly	0.700	0.653	0.526	0.000
SVM RBF	0.720	0.705	0.562	0.000
SVM Sigmoid	0.280	0.210	-0.251	0.000
Logistic Regression	0.660	0.672	0.516	0.000
Random Forest	0.680	0.717	0.482	18.641

Table 6.14
Results on the dataset with added white noise ($\sigma = 3$) using One vs Rest approach

In the context of the One vs Rest approach, the metrics of most models show a slight decrease. Currently, the QSVM with Angle encoding in the range $[0, 1]$ outperforms the linear SVM but lags slightly behind SVMs with polynomial and RBF kernels. It is worth noting that in both cases, the QSVM with Angle encoding has higher metrics than both the logistic regression model and the Random Forest.

To further investigate the performance of QSVM in managing noise during class separation, the standard deviation of the added noise is increased from $\sigma = 3$ to $\sigma = 5$. This adjustment aims to evaluate how QSVM performs under increased noise conditions.

Figure 6.11
Graphical representation of the first two principal components of the dataset with added white noise ($\sigma = 5$)



One vs One approach

Method	Accuracy	F1 Score	Matthews Correlation	Time(s)
QSVM Amplitude Encoding	0.620	0.616	0.459	238.625
QSVM Angle Encoding [0,1]	0.740	0.723	0.630	129.625
QSVM Angle Encoding [0,4π]	0.320	0.311	0.000	130.281
QSVM ZZ [0,1]	0.240	0.206	-0.108	2062.181
QSVM ZZ [0,3]	0.300	0.267	-0.031	2023.437
QSVM Qiskit	0.460	0.459	0.239	2232.815
SVM Linear	0.620	0.591	0.474	0.000
SVM Poly	0.680	0.664	0.554	0.000
SVM RBF	0.660	0.637	0.498	0.000
SVM Sigmoid	0.200	0.185	-0.374	0.000
Logistic Regression	0.720	0.703	0.607	0.000
Random Forest	0.700	0.673	0.554	23.812

Table 6.15
Results on the dataset with added white noise (σ = 5) using One vs One approach

One vs Rest approach

Method	Accuracy	F1 Score	Matthews Correlation	Time(s)
QSVM Amplitude Encoding	0.500	0.428	0.336	552.859
QSVM Angle Encoding [0,1]	0.680	0.660	0.556	320.047
QSVM Angle Encoding [0,4π]	0.340	0.334	0.033	308.015
QSVM ZZ [0,1]	0.280	0.240	-0.063	4463.234
QSVM ZZ [0,3]	0.220	0.200	-0.144	4725.968
QSVM Qiskit	0.440	0.437	0.186	4984.734
SVM Linear	0.620	0.591	0.474	0.000
SVM Poly	0.480	0.491	0.227	0.000
SVM RBF	0.580	0.598	0.369	0.000
SVM Sigmoid	0.320	0.287	-0.02	0.000
Logistic Regression	0.720	0.703	0.607	0.000
Random Forest	0.700	0.673	0.554	23.812

Table 6.16
Results on the dataset with added white noise (σ = 5) using One vs Rest approach

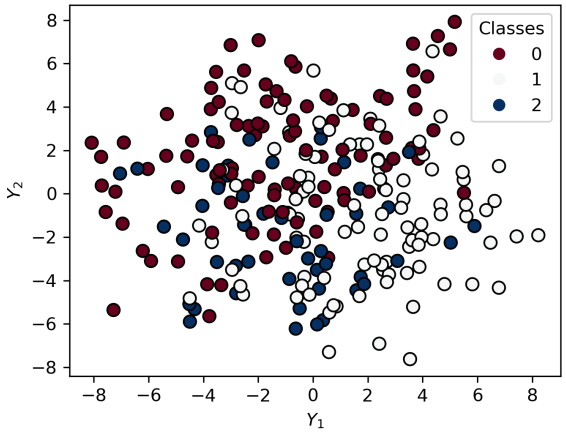
In the context of the One vs One approach, it is noteworthy that several QSVM models exhibit poor performance. The Angle encoding model in the range $[0, 1]$ has higher metrics compared to other models, including Random Forest and Logistic Regression (see Table 6.15).

Conversely, in the One vs Rest approach, Random Forest and Logistic Regression outperform the rest of the models (see Table 6.16).

As a final variation of this dataset, a similar synthetic dataset is generated but with a significantly lower standard deviation ($\sigma = 1$).

Figure 6.12

Graphical representation of the first two principal components of the dataset with added white noise ($\sigma = 1$)



One vs one approach

Method	Accuracy	F1 Score	Matthews Correlation	Time(s)
QSVM Amplitude Encoding	0.760	0.751	0.646	270.531
QSVM Angle Encoding [0,1]	0.900	0.896	0.846	156.000
QSVM Angle Encoding [0,4π]	0.380	0.277	-0.047	156.578
QSVM ZZ [0,1]	0.540	0.451	0.236	2038.156
QSVM ZZ [0,3]	0.400	0.296	-0.004	2319.625
QSVM Qiskit	0.520	0.381	0.198	2482.406
SVM Linear	0.860	0.850	0.787	0.000
SVM Poly	0.920	0.916	0.878	0.000
SVM RBF	0.840	0.839	0.761	0.000
SVM Sigmoid	0.380	0.183	0.000	0.000
Logistic Regression	0.960	0.961	0.939	0.000
Random Forest	0.780	0.762	0.668	21.984

Table 6.17

Results on the dataset with added white noise ($\sigma = 1$) using One vs One approach

Similarly to the other sigma values, the QSVM that performs best is the Angle Encoding within the normalization range $[0, 1]$, which is surpassed only by the SVM with a polynomial kernel and the Logistic Regression model, both of which demonstrate high performance (see Table 6.17).

One vs Rest approach

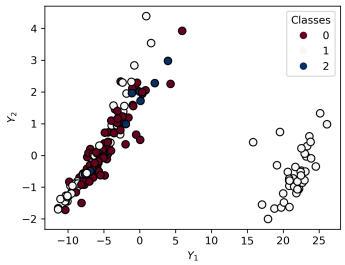
Method	Accuracy	F1 Score	Matthews Correlation	Time(s)
QSVM Amplitude Encoding	0.780	0.747	0.650	625.890
QSVM Angle Encoding [0,1]	0.920	0.914	0.875	332.031
QSVM Angle Encoding [0,4π]	0.460	0.391	0.104	331.75
QSVM ZZ [0,1]	0.500	0.419	0.181	4999.0312
QSVM ZZ [0,3]	0.420	0.314	0.037	4979.968
QSVM Qiskit	0.500	0.382	0.226	4936.281
SVM Linear	0.900	0.889	0.839	0.000
SVM Poly	0.820	0.786	0.706	0.000
SVM RBF	0.880	0.876	0.804	0.000
SVM Sigmoid	0.200	0.185	-0.374	0.000
Logistic Regression	0.960	0.961	0.939	0.000
Random Forest	0.780	0.762	0.668	21.984

Table 6.18
Results on the dataset with added white noise ($\sigma = 1$) using One vs Rest approach

Interestingly, in the One vs Rest approach, some SVMs and QSVMs have higher metrics than in the One vs One approach. In fact, for the Angle encoding in the range (0,1), both accuracy and F1 score or Matthews correlation increase, being surpassed now only by logistic regression (see Table 6.18).

As an example of multiclass classification tasks with high dimensions, the fruit dataset is used. It consists of spectra from three different cultivars of cantaloupe (Cucumis melo L. Cantaloupensis group), originally obtained from Colin Greensill at the Faculty of Engineering and Physical Systems, Central Queensland University, Rockhampton, Australia. The dataset initially comprised 1096 observations and 256 variables, but for computational feasibility, it has been reduced to 200 observations. Given the dataset's high dimensionality, only the Amplitude encoding QSVM mapping can be trained without dimensionality reduction techniques. Other Quantum Feature Maps' performance will be evaluated using PCA, with the number of components ranging from 2 to 13, approaching the simulators' maximum qubit capacity.

Figure 6.13
Graphical representation of the first 2 principal components of the Fruit's dataset



One vs one approach

Method	Accuracy	F1 Score	Matthews Correlation	Time(s)
QSVM Amplitude Encoding	0.975	0.658	0.953	17625.078
QSVM Angle Encoding [0,1]	0.950	0.641	0.903	53.296
QSVM Angle Encoding [0,4π]	0.700	0.478	0.428	55.156
QSVM ZZ [0,1]	0.550	0.371	0.121	252.875
QSVM ZZ [0,3]	0.525	0.328	0.072	330.875
QSVM Qiskit	0.550	0.371	0.121	386.152
SVM Linear	0.975	0.658	0.953	0.000
SVM Poly	0.750	0.497	0.370	0.000
SVM RBF	0.950	0.641	0.903	0.000
SVM Sigmoid	0.375	0.253	-0.218	0.000
Logistic Regression	1.000	1.000	1.000	1.000
Random Forest	0.975	0.982	0.953	25.390

Table 6.19
Results on the Fruit's dataset using One vs One approach

Method	$n_{\text{component}}$	Explained Variance Ratio
QSVM Amplitude Encoding	-	-
QSVM Angle Encoding [0,1]	4	0.990
QSVM Angle Encoding [0,4π]	5	0.992
QSVM ZZ [0,1]	4	0.990
QSVM ZZ [0,3]	4	0.990
QSVM Qiskit	4	0.990

Table 6.20
Optimal number of principal components and the corresponding explained variance ratio for each algorithm in the Fruit's dataset using One vs One approach

In the case of Quantum-Supported Vector Machine (QSVM) models, the ZZ maps do not perform well. The best QSVM model is the Amplitude Encoding one, which is the only model that can be trained without applying dimension reduction techniques. However, it is matched by the linear SVM, which has the highest metrics among the SVMs. Both of these models are surpassed by Random Forest, which has a higher F1 score due to correctly predicting the only instance of class 2 in the test, and Logistic Regression, which achieves a 100% in all metrics (see Table 6.19).

Additionally, from Table 6.22, it is observed that for the quantum models, the number of principal components that yield the highest metrics ranges between 4 and 5, with the percentage of variability explained by these components being nearly 100%.

One vs Rest approach

Method	Accuracy	F1 Score	Matthews Correlation	Time(s)
QSVM Amplitude Encoding	0.975	0.658	0.953	37437.859
QSVM Angle Encoding [0,1]	0.950	0.641	0.903	141.984
QSVM Angle Encoding [0,4π]	0.700	0.468	0.428	121.828
QSVM ZZ [0,1]	0.550	0.368	0.119	732.859
QSVM ZZ [0,3]	0.500	0.319	0.039	414.296
QSVM Qiskit	0.550	0.368	0.119	751.985
SVM Linear	0.975	0.658	0.953	0.000
SVM Poly	0.775	0.514	0.602	0.000
SVM RBF	0.950	0.641	0.903	0.000
SVM Sigmoid	0.350	0.239	-0.239	0.000
Logistic Regression	1.000	1.000	1.000	1.000
Random Forest	0.975	0.982	0.953	25.390

Table 6.21
Results on the Fruit's dataset using One vs Rest approach

Method	$n_{\text{component}}$	Explained Variance Ratio
QSVM Amplitude Encoding	-	-
QSVM Angle Encoding [0,1]	4	0.990
QSVM Angle Encoding [0,4π]	4	0.990
QSVM ZZ [0,1]	4	0.990
QSVM ZZ [0,3]	4	0.990
QSVM Qiskit	4	0.990

Table 6.22
Optimal number of principal components and the corresponding explained variance ratio for each algorithm in the Fruit's dataset using One vs Rest approach

As shown in Tables 6.21 and 6.22, there is practically no variation in the metrics of each model or in the optimal number of principal components. Since the logistic regression model achieves 100% accuracy on this dataset, it may be due to the classes being easily linearly separable. This results in a minimal difference between the One-vs-One and One-vs-Rest methods in the SVM models.

7. DISCUSSION OF THE RESULTS AND CONCLUSIONS

In this Master's thesis, we have introduced the fundamental principles of Quantum Mechanics, as well as Quantum Computing, and the key elements of basic quantum circuits. This foundation has been used to explain the theory behind Quantum Support Vector Machines (QSVMs) and various quantum kernels.

The primary objective of this work was to explore the application of quantum entanglement properties in constructing various quantum circuits. These circuits were used as kernels in Quantum Support Vector Machine (QSVM) models. The research focused on determining whether these quantum models offer any advantages in terms of performance and metrics compared to traditional Machine Learning algorithms. To this end, a considerable number of synthetic datasets with different characteristics were used to test these models in both binary and multiclass classification tasks, and to compare them with other classical algorithms in terms of execution times and other metrics of interest.

The first thing that stands out is that, despite the datasets having a small number of observations and generally a small number of predictors, the average execution time of the QSVM models was significantly high. In contrast, the execution time of the classical algorithms used was practically zero, accurate to three decimal places. The long execution times of these quantum algorithms make it challenging to analyze and compare these models, as more exhaustive analyses would require simulations for each dataset. These simulations would then be used to obtain statistics on various metrics and execution times. The two primary causes of the long execution times are:

1. **Quantum Circuit Depth**: Usually, the quantum circuits implemented in Quantum Kernels for QSVM models can be very complex and require a good deal of quantum gates, even for small datasets. Each operation made by each gate must be simulated, adding to the computational load. The depth and complexity of these circuits can significantly slow down computations.

2. **Hardware Limitations:** Although `Qiskit` and `Pennylane` are quantum software specially designed to interface with actual quantum hardware, simulating quantum computations on classical hardware is constrained by available computational power and memory. These limitations can lead to considerable delays, particularly for complex quantum algorithms.

Nevertheless, it is expected that these limitations will continue to improve, and execution times will decrease as these software programs continue to develop, which indeed they are doing at a rapid pace.

Another major limitation is the maximum number of qubits that can be included in the `Qiskit` and `Pennylane` libraries, significantly restricting the number of predictor variables that can be considered for certain quantum kernels such as Angle Encoding and ZZ maps without using dimensionality reduction techniques. This limitation arises because simulating quantum circuits classically requires substantial computational resources, as a quantum state with n qubits requires representing 2^n complex numbers. This overhead increases exponentially with the number of qubits and operations.

Setting aside execution times and the maximum number of qubits that can be used, another aspect that stands out is the difference in metric values in quantum kernels of Angle Encoding and ZZ maps for different normalization ranges, especially in Angle Encoding. This suggests that in future projects, particularly when time constraints are not an issue, the normalization range of the predictor variables should be considered as a hyperparameter and the optimal range should be determined.

Finally, regarding the performance results, it is observed that there is significant variability among the different quantum kernels and for each type of quantum kernel, performance depends on the normalization range of their predictor variables and, in the case of the ZZ map, on whether it is implemented in `Qiskit` or `Pennylane`. In general, it has been observed that for the majority of datasets, except for the latest ones, the Amplitude Encoding quantum kernel performs considerably worse than the others, despite its advantage of using a quantum circuit with n qubits for a dataset with 2^n variables. After applying these models to the entire set of datasets, no clear advantage was found for binary and multiclass classification tasks, especially compared to the Random Forest algorithm. The only datasets where any QSVM outperformed Random Forest were the binary classification dataset generated with a quantum circuit and the multiclass dataset where white noise with a standard deviation of $\sigma = 5$ was added to separate the classes. Considering these results, it would be interesting to further explore datasets with these characteristics in future analyses to see if this generally holds. Nevertheless, despite Random Forest outperforming the other models in most datasets, there are many cases where a quantum kernel outperforms basic kernel SVMs.

In conclusion, QSVM models currently face significant technical limitations, such as excessively long execution times and constraints on the maximum number of qubits they can handle. These limitations hinder the analysis of datasets with large numbers of observations and variables (big data), which are of primary interest today. Despite these challenges, and setting aside execution time considerations, QSVMs do not demonstrate a clear advantage over classical machine learning algorithms like Random Forest. However, they do show superiority over other algorithms such as SVMs with classical kernels in certain scenarios. Similar to SVMs, the performance of QSVM models varies depending on the quantum kernel used, highlighting the importance of comparing different kernels across different datasets.

Furthermore, it has been observed that certain quantum kernels, such as Angle Encoding and ZZ maps, exhibit sensitivity to the range of predictor variables, suggesting the introduction of variable range normalization as a new hyperparameter alongside the traditional C parameter. Notably, hyperparameter tuning was not extensively explored in this analysis due to its potential to significantly increase algorithm execution times.

As quantum software tools like `Qiskit` and `Pennylane` continue to evolve and execution times decrease, future research can expand these analyses to encompass high-dimensional datasets. This evolution will facilitate the comparison and identification of quantum circuits suitable for use as kernels, potentially offering advantages over classical machine learning algorithms.

BIBLIOGRAPHY

[1] E. F. Combarro and S. González-Castillo, *A practical guide to Quantum Machine Learning and Quantum Optimization*. Packt, 2023.

[2] M. Schulda, I. Sinayskiy, and F. Petruccione, *An introduction to quantum machine learning*, Quantum Research Group, School of Chemistry and Physics, University of KwaZulu-Natal, Durban, KwaZulu-Natal, 4001, South Africa, National Institute for Theoretical Physics (NITheP), KwaZulu-Natal, 4001, South Africa, September 11, 2014.

[3] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin, *Learning from Data*. AML-Book, 2

[4] M. Schuld, *Supervised quantum machine learning models are kernel methods*, Xanadu, Toronto, ON, M5G 2C8, Canada, 2021.

[5] M. Schuld and F. Petruccione, *Machine Learning with Quantum Computers*, Quantum Science and Technology, Springer International Publishing, 2021.

[6] S. Sim, P. D. Johnson, and A. Aspuru-Guzik, *Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms*, Advanced Quantum Technologies, vol. 2, no. 12, p. 1 900 070, 2019.

[7] D. Chicco and G. Jurman, *The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation*, BMC Genomics 21, 2020.

[8] G. Jurman, S. Riccadonna, and C. Furlanello,*A Comparison of MCC and CEN Error Measures in Multi-Class Prediction*, PLoS ONE **7**(8): e41882 (2012).

[9] E. Martin, *Multiclass Classification Using Support Vector Machines*. Baeldung, 2021.
https://www.baeldung.com/cs/svm-multiclass-classification.

[10] P. Fänti and S. Sieranoja, *K-means properties on six clustering benchmark datasets*, *Applied Intelligence*, vol. 48, no. 12, pp. 4743-4759, 2018.
https://doi.org/10.1007/s10489-018-1238-7

APPENDIX

Implementing the ZZ map in Qiskit

Implementing a QSVM model in Qiskit is quite simple using the class `ZZFeatureMap` from the library `quantum_machine_learning`. This class contains three main parameters:

1. **feature_dimension:** This parameter specifies the number of qubits used to encode the input data or problem size. In the ZZ feature Map, it corresponds to the number of the predictor variables p.

2. **reps:** This parameter stands for repetitions or layers in the quantum circuit. The number of reps n indicates that the quantum feature map consists of n repeated blocks or layers. Each layer typically includes a set of quantum gates designed to perform computations on the input data encoded in the qubits. For this particular case, it has been used a number of reps equal to 3. By increasing the number of reps, the quantum circuit becomes more complex, thus the QSVM model will require a longer training time.

3. **entanglement:** It specifies how qubits are entangled within each layer of the quantum circuit. In this case, it has been selected the option 'linear' entanglement, which means that each qubit is entangled with its neighboring qubits in sequence. Some other entanglement options are: 'full entanglement', 'circular Entanglement', 'sparsely entanglement', etc.

   In general, both the number of repetitions and the type of entanglement are hyperparameters that should be adjusted when implementing QSVM models in `Qiskit`.

   Qiskit Machine Learning introduces various computational tools, among them Quantum Kernels, which play a pivotal role in quantum machine learning applications such as classification and regression. One particularly significant component is the `FidelityQuantumKernel` class. This kernel class is designed to compute kernel matrices directly from datasets, facilitating rapid prototyping of models even for users with limited expertise in quantum computing.

   Here is an example of code in Python where a QSVM model is trained using the ZZ map from Qiskit using `FidelityQuantumKernel` class:

```
from qiskit.circuit.library import ZZFeatureMap
from qiskit_machine_learning.kernels import FidelityQuantumKernel
from qiskit.primitives import Sampler
from qiskit_algorithms.state_fidelities import ComputeUncompute
```

```
#number of variables in the dataset
p = X_train.shape[1]
#define the feature map that we are going to use
feature_map = ZZFeatureMap(feature_dimension = p, reps = 3,
entanglement = 'linear')
#draw the circuit in qiskit
#Apply the algorithm
sampler = Sampler()
fidelity = ComputeUncompute(sampler=sampler)
quantum_kernel = FidelityQuantumKernel(fidelity = fidelity,

feature_map = feature_map)
quantum_svc = SVC(kernel = quantum_kernel.evaluate)
quantum_svc.fit(X_train, y_train)
```

Additionally, in this code, `Sampler()` is used to simulate the execution of the quantum circuit and collect the probabilities of different measurement outcomes. The `ComputeUncompute()` function calculates the fidelity between quantum states by executing the quantum circuit and obtaining the necessary results using the sampler. More details cab ne found in https://qiskit-community.github.io/qiskit-machine-learning/.

7.0.1. Implementing the QSVM on Pennylane

As detailed in the Methodology section, most QSVM models are implemented in Pennylane, a library specifically focused on Quantum Machine Learning. Pennylane offers a straightforward approach to defining Quantum kernels and implementing them in `scikit-learn` [1], particularly for Amplitude and Angle Encoding. In all cases, the code starts as follows:

```
import pennylane as qml
nqubits = 4
dev = qml.device("lightning.qubit", wires = nqubits)
```

In the first line, the library `Pennylane` is imported with the alias 'qml'. Next, the number of qubits for the circuit is defined. In this particular example, the number of qubits is set to 4. However, in general, depending on the Quantum Kernel, it will be equal to the number of predictor variables p for Angle Encoding and ZZ Map, or $\log_2 \lceil p \rceil$ for Amplitude Encoding. Finally, a quantum device is established using `qml.device`. A quantum device is a physical or simulated system used to perform quantum computations. In this case, the computations are carried out using the simulator 'lightning.qubit', a high-performance simulator capable of handling the specified number of qubits, provided it does not exceed the software's qubit limit.

Additionally, for each Quantum Kernel, the quantum circuit that computes the kernel (4.16) has to be defined using a function.

1. **Amplitude Encoding**: In the Amplitude Encoding, the function that computes (4.16) is implemented as follows [1]:

```
@qml.qnode(dev)
def kernel_circ(x, y):
  qml.AmplitudeEmbedding(
  x, wires=range(nqubits), pad_with=0, normalize=True)
  qml.adjoint(qml.AmplitudeEmbedding(
  y, wires=range(nqubits), pad_with=0, normalize=True))
  return qml.probs(wires = range(nqubits))
```

   The decorator `@qml.qnode(dev)` must be applied before defining any quantum circuit. It transforms the function into a quantum node that is executable on the specified quantum device. The key function here is `AmplitudeEmbedding()`, which maps any vector $\vec{x}$ into the quantum state (4.18). Additionally, the function `Adjoint()`, computes the adjoint of that quantum state. Finally, with `qml.probs()`, it is estimated the probability that after applying the quantum gates $\Phi^{\dagger}(\vec{x})\Phi(\vec{y})|0\rangle$, all the qubits are in the individual quantum state $|0\rangle$, which is an estimation of (4.18). The command `pad_width=0` ensures that when 2^{nqubit} is greater than the number of predictor variables, the basis vectors of the quantum state (4.18) that are not used, are filled with zeros. Additionally, `normalize = True` ensures that the resulting quantum state is normalized. Once the function that computes the quantum kernel is defined, a matrix is constructed where each element represents the value of the quantum kernel between every pair of rows in the dataset. This matrix is then passed as an argument to the function `SVC()`, where the Support Vector Classifier (SVC) model is trained using the quantum kernel. This step remains consistent across all quantum mappings.

```
from sklearn.svm import SVC
def qkernel(X, Y):
  return np.array([[kernel_circ(x, y)[0] for y in Y] for x in X])

svm = SVC(kernel = qkernel).fit(X_train, y_train)
```

2. **Angle Encoding:** For Angle Encoding, the process of defining the quantum kernel and training the QSVM model follows the same steps as described previously, but utilizes the `AngleEmbedding()` function when implementing the quantum circuit that computes the kernel [1]:

```
def kernel_circ(x, y):
qml.AngleEmbedding(x, wires=range(nqubits))
qml.adjoint(qml.AngleEmbedding(y, wires=range(nqubits)))
return qml.probs(wires = range(nqubits))
```

3. **ZZ Feature Map:** Finally, in the ZZ map, unlike the kernels of Amplitude Encoding and Angle Encoding, there is no automatic implementation function provided by `Pennylane`. It requires manual implementation following the steps described on page 18. Bearing in mind those steps, the function that applies the ZZ Map to a row $\vec{x}$ of the dataset can be defined as follows [1]:

```
from itertools import combinations
def ZZFeatureMap(nqubits, data):
  for i in range(nqubits):
    qml.Hadamard(i)
    qml.RZ(2.0 * data[i], wires = i)
  for pair in list(combinations(range(nqubits), 2)):
    q0 = pair[0]
    q1 = pair[1]
    qml.CZ(wires = [q0, q1])
    qml.RZ(2.0 * (np.pi - data[q0])*(np.pi - data[q1]), wires = q1)
    qml.CZ(wires = [q0, q1])
```

After defining the ZZ map implementation function, the kernel calculation function is defined:

```
dev = qml.device("lightning.qubit", wires = nqubits)
@qml.qnode(dev)
def kernel_circ(x, y):
  ZZFeatureMap(nqubits, x)
  qml.adjoint(ZZFeatureMap)(nqubits, y)
  return qml.probs(wires = range(nqubits))
```