Universidad Autónoma de Madrid
Escuela politécnica superior



Máster Universitario en Ciencia de Datos

Trabajo Fin de Máster

# ESTUDIO DE ALGORITMOS DE APRENDIZAJE AUTOMÁTICO BASADOS EN COMPUTACIÓN CUÁNTICA.

Autor: Serrano Molinero, Pablo
Tutor: Gómez Arribas, Francisco Javier
Tutor: De Pedro Sánchez, Luis

Junio 2023

# ESTUDIO DE ALGORITMOS DE APRENDIZAJE AUTOMÁTICO BASADOS EN COMPUTACIÓN CUÁNTICA.

Autor: Serrano Molinero, Pablo
Tutor: Gómez Arribas, Francisco Javier
Tutor: De Pedro Sánchez, Luis

High Performance Computing and Networking Research Group
Dpto. de Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio 2023

# Resumen

En los últimos años, la computación cuántica ha surgido como un campo prometedor con el potencial de mejorar significativamente la capacidad de procesamiento y el rendimiento de los algoritmos de aprendizaje automático. En este Trabajo de Fin de Máster, se desarrolla con el objetivo de explorar el uso de máquinas de vectores de soporte cuánticas para resolver problemas de clasificación de manera más eficiente que las máquinas de vectores de soporte clásicas.

El enfoque principal de este trabajo se centra en la implementación de máquinas de vectores de soporte utilizando kernels obtenidos de manera cuántica. Nos basaremos en el problema discutido en [1] para generar distintos sets de datos que evaluaremos en nuestros clasificadores. Se llevará a cabo una comparación exhaustiva entre los resultados obtenidos con estos algoritmos cuánticos y los resultados obtenidos utilizando las máquinas de vectores de soporte clásicas correspondientes. A través de esta comparación, se demostrará el potencial de las máquinas de vectores de soporte cuánticas como herramientas poderosas en la clasificación de datos.

Los resultados obtenidos en este estudio respaldan la relevancia y el potencial de las máquinas de vectores de soporte cuánticas en la resolución eficiente de problemas de clasificación. Estos hallazgos sientan las bases para futuras investigaciones en este emocionante campo, abriendo oportunidades para explorar aplicaciones adicionales, así como para mejorar y optimizar la implementación de estos algoritmos cuánticos en problemas del mundo real.

En resumen, este Trabajo de Fin de Máster destaca la importancia de las máquinas de vectores de soporte cuánticas como herramientas prometedoras en la clasificación eficiente de datos. A través de su estudio y análisis, se establece una base sólida para futuras investigaciones y avances en este emocionante campo de la computación cuántica y el aprendizaje automático.

# Abstract

Este Trabajo de Fin de Máster explora el uso de máquinas de vectores de soporte cuánticas para la clasificación eficiente de datos. Se implementaron máquinas de vectores de soporte con kernels cuánticos y se compararon los resultados con las contrapartes clásicas. Los hallazgos demuestran el potencial de las máquinas de vectores de soporte cuánticas como herramientas poderosas en la clasificación de datos, sentando las bases para futuras investigaciones y avances en este emocionante campo de la computación cuántica y el aprendizaje automático.

# Agradecimientos

*Pablo Serrano Molinero*

*Junio 2023*

Me gustaría aprovechar esta oportunidad para expresar mi más sincero agradecimiento a todas las personas que han contribuido de manera significativa en la realización de este Trabajo de Fin de Máster.

En primer lugar, quiero agradecer a mis tutores, Francisco Gómez y Luís de Pedro, por su orientación experta y valiosos comentarios a lo largo de todo el proceso de investigación. Su dedicación, conocimientos y apoyo han sido fundamentales para el éxito de este trabajo.

Además, deseo agradecer a mis amigos y seres queridos por su constante apoyo. Sus palabras de ánimo y confianza en mí han sido un motor inspirador durante todo el proceso.

También me gustaría reconocer a la UAM, por proporcionar el acceso a los recursos necesarios para llevar a cabo este trabajo de investigación. Su colaboración ha sido fundamentales para obtener resultados significativos.

Por último, quiero expresar mi gratitud a todos los profesores, compañeros de clase y personal de la universidad que han contribuido de diversas formas a mi formación académica a lo largo de este máster. Su dedicación y enseñanzas han sido fundamentales para mi crecimiento personal y profesional.

A todos y cada uno de ustedes, ¡muchas gracias por su inestimable apoyo y por ser parte de este importante logro académico!

# Índice general

# Índice de Figuras

# Índice de Tablas

# Glosario de acrónimos

- **CLOPS**: Circuit Layer Operations per Second
- **FN**: False Negative
- **FP**: False Positive
- **QKA**: Quantum Kernel Alignment
- **QKE**: Quantum Kernel Estimation
- **QV**: Quantum Volume
- **SPSA**: Simultaneous Perturbation Stochastic Approximation
- **SVM**: Máquina de Vectores de Soporte
- **TFM**: Trabajo de Fin de Máster
- **TN**: True Negative
- **TP**: True Prositive

# 1 Introducción

Dentro del área de la ciencia de datos, el *Machine Learning* es un campo que tiene como objetivo el desarrollo de algoritmos capaces de aprender por sí mismos a través de la experiencia y el uso de datos de entrenamiento.

Debido a la ingente cantidad de información que se genera en la actualidad y la necesidad de analizarla de manera eficiente, el *Machine Learning* es una de las tecnologías con mayor desarrollo en la última década y se ha vuelto cada vez más relevante como método de clasificación, predicción y extracción de características [3].

Es habitual dividir los algoritmos de *Machine Learning* en las siguientes categorías:

- **Aprendizaje supervisado**: este tipo de algoritmos tienen como entrada un set de datos que contiene un cierto número de variables conocidas como características, así como la variable que queremos predecir. El objetivo de este tipo de algoritmos es hallar la relación que existe entre las características y la variable a predecir, de esta forma podremos usar dicha relación para predecir la variable objetivo basándonos en sus características asociadas.
- **Aprendizaje no supervisado**: a diferencia del caso anterior, este tipo de algoritmos no cuentan con una variable a predecir. Únicamente trabajan sobre las características de un set de datos con el objetivo de detectar patrones o relaciones que a priori son desconocidas.
- **Aprendizaje por refuerzo**: en este tipo de algoritmos, el computador es expuesto a un entorno en el que puede realizar acciones. Estas acciones serán recompensadas o penalizadas en función si se acerca o no a su objetivo. En este proceso la computadora va aprendiendo basándose en su propia experiencia para conseguir las máximas recompensas. Estos algoritmos son muy utilizados en bots de videojuegos o en conducción autónoma.

Dentro del aprendizaje supervisado existen dos tipos principales de problemas que podemos abordar:

- **Clasificación**: buscamos identificar la categoría a la que pertenecen nuevos datos. Por ejemplo, detección de fraude o detección de correo no deseado.
- **Regresión**: buscamos hacer una predicción de una variable numérica. Por ejemplo, probabilidad de compra o scoring de crédito.

En este TFM nos centraremos en la resolución de un problema de clasificación mediante algoritmos de *Machine Learning*.

## 1.1. Motivación

Existen numerosos métodos para abordar este tipo de problemas dentro de la computación clásica, sin embargo, en este TFM estudiaremos el uso de computadores cuánticos. El motivo detrás de esta elección es que existen algoritmos optimizados para este tipo de sistemas que han demostrado ser capaces de resolver ciertos problemas de clasificación de manera más eficiente que su contraparte clásica [4].

La computación cuántica es un paradigma distinto al de la computación clásica [5]. Su unidad fundamental es el qubit, una representación cuántica del bit clásico que aprovecha las propiedades de los estados cuánticos, como el entrelazado o la superposición. Esto hace que, por ejemplo, los qubits, gracias a la superposición cuántica, en vez de tener únicamente los estados $|0\rangle$ y $|1\rangle$ como es el caso de un bit clásico, también puedan encontrarse en una combinación lineal de ambos de la forma: $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ donde $\|\alpha\|^2 + \|\beta\|^2 = 1$.

Es el hecho de aprovechar las propiedades especiales de los estados cuánticos el que hace interesante este nuevo paradigma. Aprovechando la capacidad computacional de este tipo de sistemas, se han desarrollado algoritmos capaces de realizar operaciones de manera más eficiente que su contraparte clásica, como por ejemplo el algoritmo de Deutsch–Jozsa [6].

En la actualidad, uno de los focos de desarrollo principales de este tipo de tecnología son los algoritmos de aprendizaje automático. En este TFM nos centraremos en los conocidos como métodos de kernel, este tipo de métodos son muy utilizados en clasificadores como máquinas de vectores de soporte (SVM).

## 1.2. Objetivos

Con esta motivación, el primer objetivo de nuestro estudio consiste en desarrollar un flujo de trabajo que nos permita generar y evaluar varios casos derivados del problema descrito en [1].

Este flujo de trabajo incluirá la creación de una metodología para la generación de datos, la definición del mapa de características, así como la configuración y evaluación del clasificador cuántico. Además, se desarrollará la versión clásica del clasificador utilizado para poder realizar una comparación de desempeño con el clasificador cuántico.

Una vez definido el flujo de trabajo que seguiremos para el desarrollo y evaluación de las distintas pruebas, nuestro siguiente objetivo será evaluar el rendimiento del clasificador

cuántico en comparación con su versión clásica en los diferentes casos desarrollados. Se espera que el clasificador cuántico proporcione resultados superiores a los obtenidos por su contraparte clásica para el problema abordado.

# 2

# Estado del Arte

## 2.1. Introducción

En este capítulo describiremos el funcionamiento general de los métodos de kernel aplicados a máquinas de vectores de soporte (SVM) y definiremos matemáticamente los kernels y mapas de características cuánticos. Por último, haremos una breve introducción al problema que trataremos y por qué este resulta de interés.

## 2.2. Métodos de Kernel y SVMs

En *Machine Learning* los métodos de kernel son un tipo de algoritmos muy extendido en problemas de regresión, clasificación y *clustering*. En nuestro trabajo nos enfocaremos en su aplicación en algoritmos de clasificación binaria a través de su uso en SVMs.

Un problema de clasificación consiste en un conjunto de datos $D$ con observaciones $(x_i\ , y_i)$ donde $x_i$ es el vector de características e $y_i$ la variable que nos indica a qué clase pertenece el i-ésimo vector de características. El objetivo es entrenar un clasificador $f$ capaz de predecir a qué clase pertenecen nuevas observaciones como función de sus características, es decir, para un nuevo set de datos $T$ queremos obtener $y_i = f\left(x_i\right)$.

Las SVM aprovechan los métodos de kernel para resolver problemas que no son linealmente separables. Esto lo hacen mediante el uso de un mapa de características $\Phi : \mathcal{X} \rightarrow \mathbb{R}^N$ que mapea los datos originales del espacio $\mathcal{X}$ al nuevo espacio vectorial Euclidiano de mayor dimensión $\mathbb{R}^N$. Este nuevo espacio vectorial $\mathbb{R}^N$ debe tener definido un producto escalar $\langle \cdot\ ,\ \cdot \rangle_{\mathbb{R}^N}$ al cual de ahora en adelante conoceremos como espacio de características.

Una vez definido este mapa de características $\Phi$ definiremos la función de decisión de nuestro clasificador.

$$f(x) \; = \; sign(\langle \omega, \Phi(x) \rangle_{\mathbb{R}^N} \; + \; b)) \tag{2.1}$$

Donde el vector $\omega \in \; \mathbb{R}^N$ define el hiperplano de la frontera de decisión de nuestro clasificador y $b \in \mathbb{R}$ es un término de sesgo. Para llegar a esta función de decisión debemos entrenar nuestro clasificador para encontrar el hiperplano $\omega$, para ello, haremos que el optimizador maximice el margen. El margen se define como la mínima distancia entre nuestra frontera de decisión $\omega$ y el punto de entrenamiento más cercano. De esta forma, cuanto mayor sea el margen, más robusto será nuestro modelo a posibles variaciones estadísticas intra-clase. Este problema se puede expresar matemáticamente de la siguiente forma.

$$J(\omega, \xi) \; = \; C \sum_{i \, = \, 1}^{n} \xi_i \; + \; \frac{1}{2} ||\omega||^2 \tag{2.2}$$

Sujeto a las restricciones:

$$y_i(\langle \omega, \Phi(x) \rangle_{\mathbb{R}^N} \; + \; b) \; \geq \; 1 - \xi_i$$
$$\xi_i \geq 0$$

Donde la nueva variable $\xi_i$ puede tomar los siguientes valores:

- $\xi_i = 0$ para puntos correctamente clasificados que se encuentran fuera del margen.
- $0 \leq \xi_i \leq 1$ para puntos correctamente clasificados, pero que se encuentran dentro del margen
- $\xi_i > \; 1$ para puntos incorrectamente clasificados.

Esta variable $\xi_i$ nos permitirá maximizar el margen mientras penalizamos la clasificación incorrecta de datos. Por otro lado, el parámetro $C$ actúa como un peso que controla el error en entrenamiento y la complejidad de nuestro modelo. Un mayor valor de $C$ dará como resultado un modelo con error en entrenamiento más bajo y un valor menor de $C$ favorecerá a modelos más simples. Como hemos mencionado anteriormente, este problema normalmente se resuelve en un nuevo espacio de dimensionalidad mayor al original, por lo que en ocasiones esta mayor dimensionalidad hace que resolver la optimización propuesta sea extremadamente complejo. Es en este punto donde los métodos de kernel son realmente útiles. La función de kernel simplemente se define como el producto escalar de dos puntos de nuestro mapa de características.

$$K(x_i, x_j) = \; \langle \Phi(x_i), \Phi(x_i) \rangle_{\mathbb{R}^N} \tag{2.3}$$

Utilizaremos este kernel para resolver el problema planteado en su representación dual. Esta nueva formulación del problema original mediante multiplicadores de Lagrange nos permitirá hacer uso del conocido "truco del kernel".

$$\hat{L}(\alpha) = \sum_{i \, = \, 1}^{n} \alpha_i - \frac{1}{2} \sum_{i \, = \, 1}^{n} \sum_{j \, = \, 1}^{n} \alpha_i \alpha_j y_i y_j x_i x_j \tag{2.4}$$

Sujeto a las restricciones:

$$0 \leq \alpha_i \leq C$$

$$\sum_{i\,=\,1}^{n} \alpha_i y_i = 0$$

El resultado de resolver esta optimización nos dará los $\alpha_i$ óptimos que definirán nuestro vector $\omega \;=\; \sum_{i\,=\,1}^{n} \alpha_i y_i \Phi(x_i)$. De esta forma podemos reformular 2.1 en función del kernel.

$$f(x) \;=\; sign(\sum_{i\,=\,1}^{n} \alpha_i y_i K(x_i, x) \;+\; b) \tag{2.5}$$

Esta nueva forma de afrontar el problema, conocida como "truco del kernel" nos permitirá que tanto el entrenamiento de la SVM definido en 2.4 como la clasificación en 2.5 se ejecuten en un espacio arbitrario N-dimensional, inclusive en espacios de Hilbert de dimensión infinita, siempre y cuando sea posible una evaluación eficiente de la función de kernel.

## 2.3. Computadores Cuánticos

Un ordenador cuántico es un tipo de dispositivo computacional que utiliza principios de la mecánica cuántica para realizar cálculos y procesamientos de información. Estos dispositivos se caracterizan por varios parámetros clave que determinan su desempeño y capacidad de procesamiento. Estos parámetros incluyen:

- **Qubits**: Los qubits son los elementos fundamentales de un ordenador cuántico. Son análogos a los bits clásicos y representan la unidad básica de información cuántica. Los qubits tienen la capacidad de existir en múltiples estados al mismo tiempo gracias a la superposición cuántica. Cuantos más qubits tenga un ordenador cuántico, mayor será su capacidad de procesamiento y la complejidad de los problemas que puede abordar.

- **Arquitectura del procesador**: Otra de las características principales de un computador cuántico es la arquitectura de su procesador. Esta arquitectura está formada por las conexiones entre los distintos qubits que conforman el sistema.

  Estas arquitecturas están definida por el grafo de conectividad. Este grafo de conectividad es una representación gráfica de la disposición física de los qubits en el procesador, donde cada nodo representa un qubit y cada arista una conexión física entre dos qubits. Estas conexiones permitirán la interacción de los qubits entre sí.

  La estructura del grafo de conexión puede influir en la forma en que se pueden realizar operaciones entre los qubits. En algunos casos, solo se permite la interacción directa entre qubits vecinos en el grafo de conexión, lo que puede afectar al diseño y la ejecución de algoritmos cuánticos. Además, la distancia entre los qubits en el grafo de conexión puede afectar a la cantidad de operaciones necesarias para realizar ciertos cálculos y puede influir en la calidad y la escalabilidad del procesador cuántico.

  En la figura 2.1 podemos observar dos arquitecturas distintas para ordenadores cuánticos de 5 qubits.

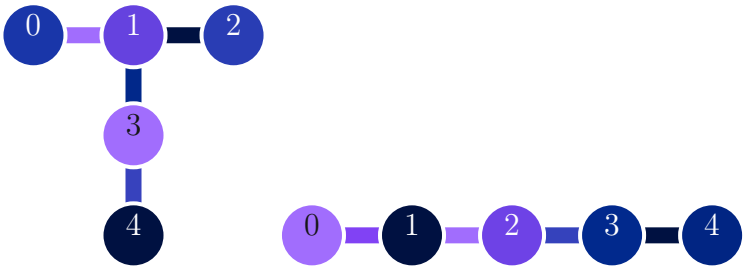


Figura 2.1: Grafos de conectividad de *IBMQ Belem* e *IBMQ Manila*.

- ***Circuit Layer Operations per Second* (CLOPS)**: Un procesador cuántico ejecuta operaciones lógicas en forma de circuitos cuánticos. Los circuitos consisten en una secuencia de puertas cuánticas aplicadas a los distintos qubits. Estas operaciones de capa de circuito pueden incluir puertas cuánticas como puertas de un solo qubit (por ejemplo, X, Y, Z o H) y puertas de dos o más qubits (por ejemplo, CNOT o CX) [7].

  Esta medida se utiliza para evaluar la capacidad de procesamiento de un procesador cuántico y se expresa en términos del número de operaciones de capa de circuito que puede realizar en un segundo. Cuanto mayores sean las CLOPS, mayor será la capacidad de procesamiento y la velocidad del procesador cuántico.

  Es importante tener en cuenta que las CLOPS es solo una medida de velocidad y no refleja necesariamente el rendimiento general de un procesador cuántico. Otros factores, como la estabilidad de los qubits, la corrección de errores y la escalabilidad, también son fundamentales para evaluar la calidad y el desempeño de un sistema cuántico.

- ***Quantum Volume* (QV)**: El QV es una métrica que se utiliza para evaluar la capacidad de computo de un ordenador cuántico de manera holística. Esta métrica se puede obtener mediante la aplicación de un protocolo [8] [9] en computadores cuánticos de un tamaño no demasiado grande. El protocolo cuantifica cual es el mayor circuito con igual anchura y profundidad que se puede ejecutar de manera fiable en el computador cuántico a evaluar. Ordenadores cuánticos con alta fidelidad, alta conectividad y un gran número de puertas calibradas suelen tener valores mayores de QV.

## 2.4. Simuladores Cuánticos

Además de ordenadores cuánticos, también existen simuladores. Un simulador de ordenador cuántico es un software diseñado para simular el comportamiento y las operaciones de un ordenador cuántico en un entorno clásico, es decir, utilizando un ordenador convencional. Aunque no tienen la capacidad de ejecutar cálculos cuánticos reales, los simuladores de ordenador cuántico permiten estudiar y analizar el funcionamiento de algoritmos y circuitos cuánticos, así como explorar las propiedades y características de sistemas cuánticos sin necesidad de tener acceso a un sistema cuántico real.

Estos simuladores se basan en algoritmos clásicos que emulan el comportamiento de los qubits y las operaciones cuánticas. Utilizan técnicas de simulación que involucran

matrices, vectores o técnicas de Monte Carlo [10] para modelar y seguir el estado cuántico de los qubits a medida que se realizan las operaciones cuánticas.

Los simuladores cuentan con ciertas desventajas al compararlos con un ordenador cuántico real. Debido a la naturaleza exponencial de los cálculos cuánticos, los simuladores pueden volverse computacionalmente costosos a medida que aumenta el número de qubits simulados. Además, los simuladores no pueden capturar completamente los efectos de ruido y error cuántico que ocurren en los sistemas reales.

Sin embargo, esta tecnología también ofrece varias ventajas. En primer lugar, son más accesibles y fáciles de usar en comparación con los ordenadores cuánticos reales, ya que no requieren hardware especializado ni acceso a laboratorios de investigación. Además, los simuladores permiten una depuración y un análisis detallado del comportamiento de los circuitos cuánticos, lo que facilita el desarrollo y la optimización de algoritmos antes de su implementación en un procesador cuántico real.

En este TFM hemos utilizado tanto ordenadores cuánticos como simuladores. La mayoría de pruebas se ejecutarán en simuladores debido a las ventajas que estos ofrecen, destacando su gran disponibilidad.

## 2.5. Kernel y Mapas de Características Cuánticos

Como ya hemos introducido anteriormente, uno de los usos de la computación cuántica es en algoritmos de *Machine Learning* como el mencionado anteriormente. En este caso estudiaremos como un modelo de clasificación SVM podría beneficiarse del uso de kernels y mapas de características cuánticos.

Para que sea posible obtener beneficios gracias al uso de ordenadores cuánticos para este tipo de cálculos, es necesario que estos no sean estimables eficientemente en ordenadores clásicos, es decir, para obtener un posible beneficio es necesario que el mapa de características que implementemos en nuestro ordenador cuántico sea muy costoso de estimar de forma clásica, siendo esto una condición necesaria pero no suficiente. Este mapa de características cuántico definirá la función de kernel que usará nuestra SVM.

De esta forma, podemos observar que la única diferencia entre una SVM convencional y una SVM que utilice un kernel cuántico es que, en el caso cuántico, será un ordenador cuántico el encargado de estimar las entradas del kernel.

Este kernel puede entenderse como la probabilidad de transición entre los estados $|0^n\rangle$ de un operador unitario que actúa sobre $n$ qubits. La aplicación del mapa de características sobre cada dato $x \in \mathcal{X}$ se puede entender como la actuación del unitario $\mathcal{U}(x)$ sobre el estado de referencia $|0^n\rangle \langle 0^n|$.

$$\Phi(x) = \mathcal{U}(x) |0^n\rangle \langle 0^n| \mathcal{U}^\dagger(x) \tag{2.6}$$

Una vez definido nuestro mapa de características definiremos nuestro kernel cuántico. Para ello primero definiremos el producto escalar entre matrices como $\langle A, B\rangle = tr(A^\dagger B)$, de esta forma el espacio de matrices Hermíticas es un espacio vectorial Euclidiano. El espacio generado de esta forma se puede mapear con $\mathbb{R}^N$ si elegimos una base adecuada,

en este caso las matrices de Pauli [5]. De esta forma podemos expresar el kernel como el producto escalar entre los estados $\Phi(x)$ de los diferentes puntos $K(x, \tilde{x}) = \langle \Phi(x), \Phi(\tilde{x}) \rangle$.

$$K(x, \tilde{x}) = |\langle 0^n | \mathcal{U}^\dagger(x) \mathcal{U}(\tilde{x}) | 0^n \rangle|^2 \tag{2.7}$$

## 2.6. Descripción del Problema y su Interés

Como hemos anticipado anteriormente, la elección de kernel cuánticos para su uso en SVMs está motivada por el beneficio que podrían aportar para cierto tipo problemas. Se ha demostrado que es posible encontrar problemas para los cuales el acceso a métodos de computación cuánticos ofrece mejores resultados que los obtenidos únicamente con métodos de computación clásica. Esta ventaja no solo ha sido estudiada de manera teórica [4], sino que también se ha implementado en ordenadores cuánticos reales, probando que la resolución de este tipo de problemas en dispositivos cuánticos ofrece resultados con precisiones cercanas al 100 % [1].

En este TFM nos centraremos en el problema descrito en [1]. Este problema se llamará *labeling cosets with error*.

Definiremos el problema para un grupo $G$ y su espacio homogéneo definido por un subgrupo $S < G$. Definiremos ahora 2 clases laterales o cosets $c_+ \ , \ c_- \in G$ tal que:

$$C_\pm = c_\pm \, S \tag{2.8}$$

La definición de las clases laterales se puede hacer mediante un sampling aleatorio de los elementos del grupo. Esto nos permitirá generar los dos grupos $\mathcal{Q}^\epsilon_\pm : \mathcal{Q} \to \mathbb{R}^+_0$ cuya masa debe estar centrada en una de las dos clases laterales $C_\pm$ que hemos definido, esto hace que los elementos puedan desviarse de la clase a la que pertenecen un factor de error $\epsilon$.

Una vez generados los grupos, nuestro clasificador binario deberá determinar a qué grupo pertenece un dato $x \in \mathcal{T}$ de nuestra muestra de test.

Para ilustrar el problema definido, veremos un ejemplo para el caso de un qubit con $G = SU(2)$ y $S = \{\mathbb{I}, A, A^2\}$ donde $A = exp(i(2\pi/3)X)$ y $X$ la matriz de Pauli $X$. En este caso elegiremos los elementos $c_+ \ , \ c_- \in G$ tal que su representación matricial sea $D_{c_+} = \mathbb{I}$ y $D_{c_-} = H$ donde $H$ representa la puerta de Haddamard.

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{2.9}$$

Esta matriz actúa de la siguiente forma sobre los estados $|0\rangle, |1\rangle$:

$$H|0\rangle = |+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \tag{2.10}$$

$$H|1\rangle = |-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \tag{2.11}$$

Formamos ahora las clases laterales haciendo uso de (3.6):

$$C_+ = D_{c_+} S = S = \{\mathbb{I}, A, A^2\} \tag{2.12}$$

$$C_- = D_{c_-} S = \{H, HA, HA^2\} \tag{2.13}$$

Con estas clases laterales generaremos puntos $x \in C_\pm$ con $x = c_\pm s$ de manera que podemos mapearlos con los estados $D_{c_-} D_s |\psi\rangle$ y $D_{c_+} D_s |\psi\rangle$ para todo $s \in S$.

En función de la elección de $|\psi\rangle$, los cosets obtenidos pueden variar.

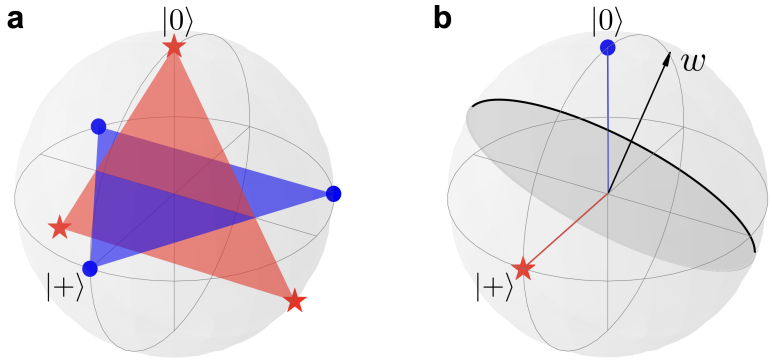


Figura 2.2: a) Cosets resultado de usar como estado fiduciario $|0\rangle$. b) Cosets resultado de utilizar como estado fiduciario $|1\rangle$. Fuente: [1]

Podemos observar en la figura 2.2a, donde el estado elegido es $|0\rangle$, que estas clases no son linealmente separables en la esfera de Bloch. Casos como este son los que se intentan resolver mediante el uso de mapas de características de mayor dimensionalidad y métodos de kernel que permitan que el problema sí sea linealmente separable en el nuevo espacio de dimensionalidad mayor.

En el caso de haber elegido el estado $|1\rangle$, los resultados sí son linealmente separables por el plano definido por $\omega$ por lo que no sería necesario resolver el problema en un espacio de dimensionalidad mayor.

Este pequeño ejemplo en un qubit es ilustrativo del tipo de problema que estamos tratando. En este TFM resolveremos una extensión del problema descrito mediante el uso de métodos clásicos y cuánticos para comparar los resultados obtenidos y estimar si, para cierto tipo de problemas, se puede obtener una ventaja cuántica.

## 2.7. Conclusiones

En este capítulo se ha descrito la aplicación de kernels a un clasificador basado en SVM, así como el formalismo matemático que hay detrás de los kernels y mapas de características cuánticos. Además, se ha presentado el problema que intentaremos resolver mediante el uso de kernels cuánticos y como estos pueden ofrecer ventajas sobre su contraparte clásica para este tipo de problemas [4].

En el siguiente capítulo describiremos más en detalle el tipo de kernels que se van a utilizar, el proceso de entrenamiento de nuestro clasificador y la estructura de los datos que utilizaremos.

# 3
# Diseño

## 3.1. Introducción

En este capitulo se describirán los pasos tomados para el desarrollo y estudio de nuestro clasificador cuántico.

- Describiremos en detalle el tipo de kernel cuántico utilizado así como el método que utilizaremos para su estimación.
- Definiremos el proceso de entrenamiento de una SVM que utiliza un kernel cuántico.
- Detallaremos el proceso de generación de datos.

## 3.2. *Covariant Quantum Kernels*

Dentro de las múltiples opciones que existen a la hora de elegir nuestro kernel cuántico, en este estudio nos centraremos en un tipo de kernels que funcionan muy bien para datasets con estructura de grupo, esto es, los datos $\mathcal{X}$ pueden ser vistos como parte de un grupo más grande $\mathcal{X} \subseteq G$, donde $G$ dependerá del problema que queramos resolver. Este tipo de kernels es conocido como *Covariant Quantum Kernels* [11] y están formados por dos elementos principales:

- **Operador unitario**: Nuestro mapa de características se basará en una transformación unitaria $D_x$ que actua sobre cada punto $x \in \mathcal{X}$ de los datos. Este operador debe ser implementable a través de un circuito de $n$-qubits que podamos evaluar en nuestro computador cuántico

- **Estado cuántico de referencia**: Para que nuestro kernel pueda ser evaluado necesitamos un estado cuántico de referencia $|\psi\rangle \in (\mathbb{C}^2)^{\otimes n}$. Parte de nuestro trabajo a la hora de definir el kernel para nuestro problema es optimizar este estado de referencia. Para ello, lo parametrizaremos a través de un circuito $V(\lambda)$ con $\lambda \subset \mathbb{R}$ que nos permitirá definir nuestro estado de referencia como $|\psi\rangle = V(\lambda)\,|0^n\rangle$.

Una vez hemos definido los dos elementos que definirán nuestro kernel podemos expresar tanto el mapa de características como el kernel como función de estos elementos.

$$K(x, z) \;=\; |\,\langle\psi|D_x^\dagger D_{\tilde{x}}|\psi\rangle\,|^2 \tag{3.1}$$

Dado que hemos definido el estado $|\psi\rangle$ como el resultado de aplicar un circuito $V(\lambda)$ sobre el estado $|0^n\rangle$ podemos redefinir el operador que actua en nuestro kernel de la siguiente forma.

$$U(x) = D_x V(\lambda) \tag{3.2}$$

En la figura 3.1 podemos observar la estructura base del circuito cuántico que utilizaremos para estimar nuestro kernel cuántico.

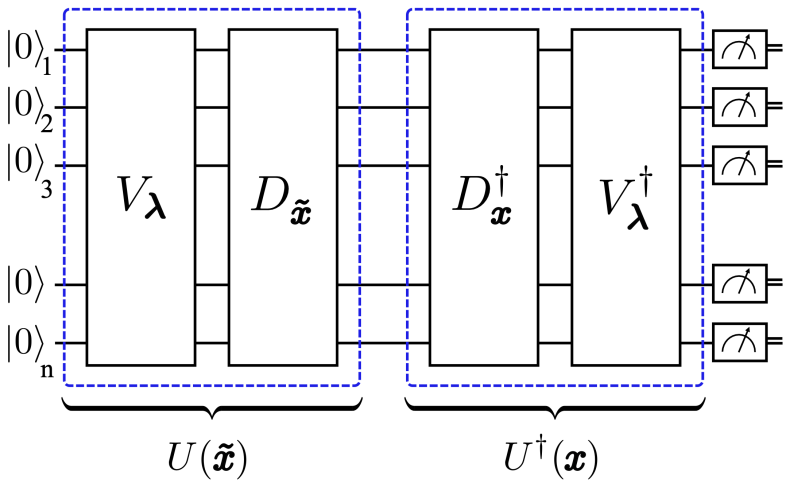


Figura 3.1: Circuito cuántico para calcular los elementos del kernel. Fuente: [1]

Por otro lado es importante definir como vamos a escoger nuestro estado de referencia. En este caso usaremos un método muy extendido en kernels clásicos conocido como *kernel alignment*. Como ya hemos mencionado anteriormente este estado de referencia $|\psi\rangle$ lo parametrizaremos mediante un circuito variacional $V(\lambda)$. De esta forma nuestro kernel $K_\lambda(x, \tilde{x})$ también quedará parametrizado.

Esto generará múltiples fronteras de decisión $f(x) \;=\; sign(\sum_{i\,=\,1}^{n} \alpha_i y_i K_\lambda(x_i, x) + b)$. En este punto debemos optimizar no solo los parámetros $\alpha_i$ sino también $\lambda$. Para ello utilizamos el método de *quantum kernel alignment*(QKA), este método nos permitirá optimizar el límite superior del error de nuestra SVM a través de la siguiente función de coste.

$$\hat{L}(\alpha) = \sum_{i\,=\,1}^{n} \alpha_i - \frac{1}{2} \sum_{i\,=\,1}^{n} \sum_{j\,=\,1}^{n} \alpha_i \alpha_j y_i y_j K_\lambda(x_i, x_j) \tag{3.3}$$

Este proceso de optimización busca un límite superior del error de generalización como función de $\alpha$. Además, el proceso de QKA minimizará este límite superior del error de generalización con respecto al parámetro $\lambda$ sujeto a las restricciones descritas en el capítulo 2.2.

$$min_\lambda max_\alpha F(\alpha, \lambda) \tag{3.4}$$

Para esto, es imprescindible que el nuevo operador $U(x)$ sea estimable a través de un circuito en un computador cuántico. Si esta premisa se cumple podemos tratar cada entrada de nuestro kernel como la probabilidad de transición entre el estado $|0^n\rangle$ evolucionado con $U^\dagger(x)U(\tilde{x})$ y el estado $|0^n\rangle$. Este método de estimación de kernels cuánticos es conocido como *Quantum Kernel Estimation* (QKE).

$$K_\lambda(x, z) \ = \ |\langle 0^n|V^\dagger(\lambda)D_x^\dagger V(\lambda)D_{\tilde{x}}|0^n\rangle|^2 \tag{3.5}$$

En la figura 3.2 podemos observar el circuito que utilizaremos para la estimación de las distintas entradas del kernel. Este circuito está definido en función de las rotaciones de un solo qubit $R_P(\lambda) = exp(-i(\lambda/2)P)$ con $P \in X, Y, Z$ y las puertas de entrelazado cuántico $CZ$ que haremos coincidir con la conectividad del procesador utilizado.

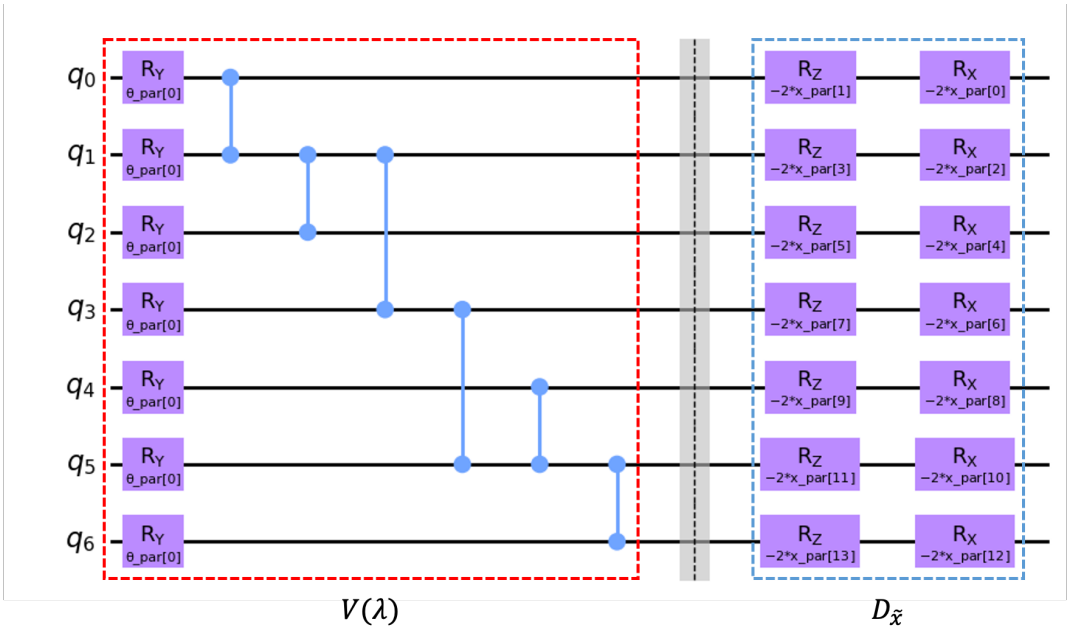


Figura 3.2: Circuito cuántico para calcular los elementos del kernel.

Una vez definidos el kernel y nuestra función de coste a optimizar, en la siguiente sección detallaremos el método de entrenamiento utilizado para nuestro clasificador.

## 3.3. Entrenamiendo del clasificador

Para el entrenamiento de nuestro clasificador haremos uso de un algoritmo estocástico de optimización conocido como SPSA [12]. Este algoritmo funciona de manera iterativa optimizando la función de coste en cada iteracion, por ello este proceso necesita tanto de un computador clásico como de uno cuántico. Para comprender mejor el funcionamiento de nuestro algoritmo vamos a detallar los diferentes pasos de sus iteraciones.

1. Entrada: Proporcionaremos como entrada nuestros datos de entrenamiento $T$.

2. Configuración del entrenamiento: Debemos configurar los parámetros del entrenamiento de nuestro clasificador. Debemos establecer el valor de nuestro parámetro de regularización $C$, el número de medidas $R$ que va a realizar nuestro ordenador cuántico, los parámetros iniciales de la optimización $\lambda_0$ y el número máximo de iteraciones $P$.

3. Configuración de nuestro computador cuántico: Debemos escoger el hardware cuántico adecuado para poder evaluar las entradas de nuestro kernel.

4. Inicialización del kernel cuántico: Estableceremos $\lambda = \lambda_0$ para el circuito que calculará las entradas de nuestro kernel.

5. Proceso de optimización: Este proceso ejecutará los siguientes pasos desde $i = 0$ hasta $P$:

   *a*) Generar un vector aleatorio $\epsilon \in -1, 1^n$ donde n es el número de qubits que utilizaremos en nuestro circuito.

   *b*) Evaluar $\lambda_{\pm,i} = \lambda_i \pm c_i\epsilon$ donde $c_i = \frac{c}{(i+1)^\gamma}$ para $\gamma, c$ constantes.

   *c*) Evaluar las matrices de kernel $K_\pm = K(\lambda_{\pm,i}, T)$ en nuestro hardware cuántico con $R$ medidas por iteración.

   *d*) Maximizamos la función de coste $F(\alpha_{\pm,i}, \lambda_{\pm,i})$ en función de $\alpha$. Este proceso se hará en nuestro ordenador clásico mediante el uso de métodos de resolución de problemas cuadráticos como CVXOPT [13].

   *e*) Actualizamos el parámetro $\lambda_{i+1} = \lambda_i - \frac{a_i}{2c_i}[F(\alpha_{+,i}, \lambda_{+,i}) - F(\alpha_{-,i}, \lambda_{-,i})]$ mediante el algoritmo SPSA con $a_i = \frac{a}{(i+1+A)^\sigma}$ para $A, \sigma$ constantes.

6. Una vez finalizado el proceso de optimización evaluaremos nuestro kernel para los parámetros finales $\lambda^*$.

## 3.4. Generación de Datos

En esta sección detallaremos la estructura y el proceso de generación de los datos $\mathcal{D}$ que utilizaremos para el entrenamiento y la posterior evaluación de nuestro modelo de clasificación cuántico.

Los datos que generaremos representarán el problema descrito en [1], denominado *labeling cosets with error*. En el artículo de referencia, se discute este problema en el contexto de un procesador cuántico de 26 qubits. A pesar de nuestros intentos por obtener acceso a los datos y al código utilizado en el artículo a través de interacciones con el equipo de IBM que lo desarrolló, no hemos logrado obtener dicha información.

Debido a esta limitación, en este TFM hemos decidido definir una metodología que nos permita generar datos para cualquier arquitectura cuántica. De esta manera, podremos evaluar distintos grados de complejidad para el problema estudiado.

La base matemática en la que nos fundamentaremos para desarrollar esta metodología de trabajo es la siguiente.

Definiremos el problema para un grupo $G$ y su espacio homogéneo definido por un subgrupo $S < G$. Definiremos ahora 2 clases laterales o cosets $c_+ \ , \ c_- \ \in \ G$ tal que:

$$C_\pm \ = \ c_\pm \ S \tag{3.6}$$

La definición de las clases laterales se puede hacer mediante un sampling aleatorio de los elementos del grupo. Esto nos permitirá generar los dos grupos $\mathcal{Q}^\epsilon_\pm : \mathcal{Q} \rightarrow \ \mathbb{R}^+_0$ cuya masa debe estar centrada en una de las dos clases laterales $C_\pm$ que hemos definido. Los datos pertenecientes a cada uno de los grupos pueden desviarse de la clase a la que pertenecen un pequeño factor de error $\epsilon$.

Una vez generados los grupos, nuestro clasificador binario deberá determinar a que grupo pertenece un dato $x \in \mathcal{T}$ de nuestra muestra de test.

Para generar los datos que representan este problema nos apoyaremos en la arquitectura del hardware cuántico en el cual entrenaremos nuestro modelo. En este ejemplo hemos elegido el computador *IBMQ Nairobi*, un computador cuántico de 7 qubits con el siguiente grafo de conectividad.

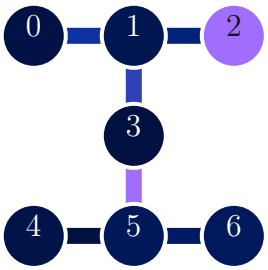


Figura 3.3: Grafo de conectividad de *IBMQ Nairobi*.

Una vez seleccionado el hardware cuántico que utilizaremos, elegiremos $G = SU(2)^{\otimes n}$ con la representación natural $D$ de $SU(2)$ para cada qubit y un grupo estabilizador de Pauli $S = \langle s_1, ..., s_i \rangle$ [14]. En particular seleccionaremos el estabilizador del grafo de conectividad de nuestro computador cuántico [15].

$$S_{graph} = \langle \{ X_i \bigotimes_{k:(k,i)\in E} Z_k \}_{i \in V} \rangle \tag{3.7}$$

Donde $E$ representan las conexiónes del procesador y $V$ los distintos qubits. Es importante destacar que para el estabilizador del grafo $S$ se puede definir un estado estabilizador $|\psi\rangle$ tal que $D_s |\psi\rangle = |\psi\rangle$. Para ese estado $|\psi\rangle$, cada dato se mapea a un estado que representa cada coset.

$$D_{c_\pm s} |\psi\rangle = D_{c_\pm} D_s |\psi\rangle = D_{c_\pm} |\psi\rangle \tag{3.8}$$

Al añadir al problema un pequeño término de error $\epsilon$, estos dos estados se ven perturbados ligeramente. Cuando el término de error es $\epsilon << 1$ esperamos que los estados perturbados sean clasificados también de manera correcta.

Para generar observaciones en nuestros datos utilizaremos la siguiente definición de rotación $D(\theta_1, \theta_2, \theta_3) = exp(-i(\theta_1/2)X)exp(-i(\theta_2/2)Z)exp(-i(\theta_3/2)X) \in SU(2)$ tal que

cada qubit queda parametrizado por sus ángulos de Euler. El hecho de que elijamos definir las rotaciones en función de los ángulos de Euler se debe a la propiedad de que los operadores unitarios especiales $A \in SU(2)$ pueden ser expresados como descomposición de rotaciones [16].

$$A = \begin{pmatrix} e^{i\theta_1/2} & 0 \\ 0 & e^{-i\theta_1/2} \end{pmatrix} \cdot \begin{pmatrix} \cos\theta_2/2 & \sin\theta_2/2 \\ -\sin\theta_2/2 & \cos\theta_2/2 \end{pmatrix} \cdot \begin{pmatrix} e^{i\theta_3/2} & 0 \\ 0 & e^{-i\theta_3/2} \end{pmatrix} \tag{3.9}$$

En nuestro caso únicamente utilizaremos los dos primeros ángulos de Euler, es decir, $\theta_3 = 0$. Para generar observaciones seleccionaremos de manera aleatoria dos sets de ángulos $\theta_\pm \in [-\pi/2, \pi/2]^{2n}$ donde $n$ es el número de qubits que utilizaremos para definir los diferentes cosets.

Una vez contamos con los ángulos $\theta_\pm$ construiremos los cosets $C_\pm = R(\theta_\pm)S_{graph}$ donde $R(\theta_\pm) = \otimes_{k=1}^{n} D(\theta_\pm^{2k-1}, \theta_\pm^{2k}, 0)$. En este punto es donde haremos uso de (3.9) para expresar $C_\pm$ como función de sus dos primeros ángulos de Euler.

$$C_\pm = R(\theta_\pm)S_{graph} = R(\phi_\pm) \tag{3.10}$$

Llegados a este punto detallaremos los cálculos necesarios para expresar $C_\pm$ en función de sus ángulos de Eulrer $(\phi_1, \phi_2)$.

De esta forma cada punto de nuestro conjunto de datos se puede expresar como una rotación $R(\phi) = \otimes_{k=1}^{n} R(\phi_\pm^{2k-1}, \phi_\pm^{2k}, 0)$ caracterizada por $2n$ ángulos de Euler. Obteniendo así observaciones con la estructura $(\phi_i, y_i)$ donde $\phi_i = (\phi_{y_i}^1, \phi_{y_i}^2, ..., \phi_{y_i}^{2n-1}, \phi_{y_i}^{2n})$ e $y_i = \pm 1$ con $y_i = 1$ si el elemento $i$ pertenece a $C_+$ e $y_i = -1$ si el elemento pertenece a $C_-$. En este proceso se generarán el mismo número de observaciones pertenecientes a $C_+$ y $C_-$.

Siguiendo estas premisas se ha desarrollado una metodología de generación de datos que nos permitirá crear sets de datos para la arquitectura de referencia escogida. Con esta metodología se han generado datos para 5, 7, 10 y 16 qubits de manera que tengamos el mismo número de observaciones de cada coset.

## 3.5. Conclusiones

En esta sección se ha descrito el tipo de kernel cuántico que utilizaremos en el estudio, conocido como *Covariant Quantum Kernel*, así como el método que se utilizará para su estimación. Tras esto, hemos detallado el proceso de entrenamiento de una SVM con un kernel cuántico. Por último hemos discutido la motivación para diseñar un proceso de generación de datos así como las bases para comprender el problema sobre el que vamos a trabajar.

En el siguiente capitulo nos enfocaremos en describir detalladamente los desarrollos realizados en base al diseño que acabamos de definir. Explicaremos los códigos desarrollados y las tecnologías utilizadas, así como los diferentes retos a los que nos hemos enfrentado a lo largo del desarrollo de este TFM.

# 4

# Desarrollo

## 4.1. Introducción

En este capítulo describiremos en detalle los desarrollos realizados en este TFM, así como las distintas tecnologías utilizadas. En nuestro estudio hemos cubierto el ciclo completo del problema descrito, desde la generación de datos en MATLAB hasta la ejecución del clasificador en un *Jupyter Notebook* de Python. En la figura 4.1 podemos observar el flujo de trabajo seguido así como las tecnologías utilizadas en cada punto.

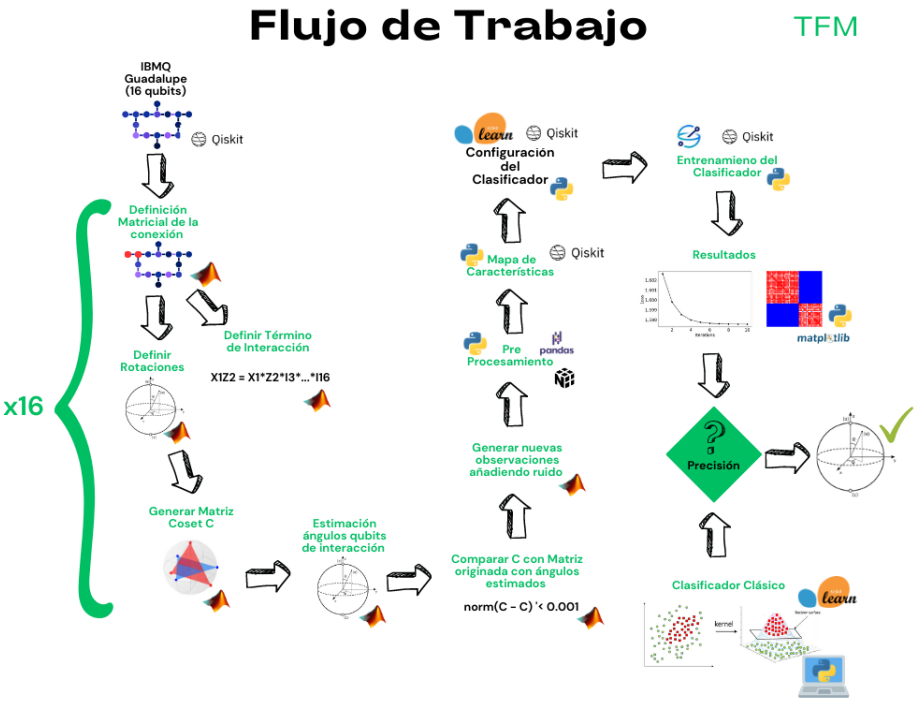


Figura 4.1: Flujo de Trabajo del TFM

## 4.2. Tecnologías

En esta sección detallaremos las distintas tecnologías utilizadas en los desarrollos realizados en nuestro estudio.

### 4.2.1. Generación de Datos

Para la generación de datos hemos utilizado MATLAB. MATLAB es un entorno de programación y lenguaje de alto nivel utilizado principalmente para el procesamiento numérico y el análisis de datos. Está diseñado para manejar eficientemente operaciones matriciales y vectoriales, proporcionando una amplia variedad de funciones y operadores optimizados para realizar cálculos en matrices grandes de manera rápida y precisa.

Esta eficiencia en los cálculos matriciales lo hace una opción muy adecuada para implementar el proceso de generación de datos detallado en **3.4**.

Pese a esto, cabe destacar que este proceso de generación de datos es muy costoso computacionalmente. Esto es debido a que las matrices generadas durante los cálculos son de un gran tamaño, por lo que las operaciones entre ellas son computacionalmente muy costoas. Tanto es así que para la obtención del set de datos de 16 qubits se necesitó de un servidor dedicado de la Escuela Politécnica de la UAM. En la tabla 4.1 podemos ver las especificaciones de dicho servidor.

| Información del Sistema | |
|---|---|
| OS | Linux |
| CPUs | 2 |
| Cores | 64 |
| Memoria[Gb] | 512 |

Tabla 4.1: Tabla de especificaciones del servidor de la Escuela Politécnica de la UAM.

### 4.2.2. Computación Cuántica

Para realizar cálculos cuánticos utilizaremos las distintas herramientas disponibles en *IBM Quantum*. Dentro de *IBM Quamtum* utilizaremos las siguientes tecnologías:

#### Qiskit

Qiskit es una biblioteca de código abierto desarrollada por IBM para trabajar con computación cuántica. Es una de las plataformas más populares y ampliamente utilizadas para la programación y el desarrollo de algoritmos cuánticos.

Qiskit proporciona un conjunto completo de herramientas y bibliotecas para trabajar con qubits, circuitos cuánticos y distintos tipos de sistemas cuánticos. Está desarrollado en Python y ofrece una interfaz de programación fácil de usar que permite diseñar, simular y ejecutar circuitos cuánticos en simuladores o en dispositivos reales ofrecidos por *IBM Quantum Experience*.

Utilizaremos la API de Qiskit para la implementación de los clasificadores cuánticos que queremos estudiar, así como para conectarnos a los distintos recursos de *IBM Quantum Experience*.

En la tabla 4.2 podemos ver las versiones de los diferentes paquetes del softawe de Qiskit que hemos utilizado en este estudio.

| Paquete | Version |
|---|---|
| qiskit-terra | 0.24.0 |
| qiskit-aer | 0.12.0 |
| qiskit-ibmq-provider | 0.20.2 |
| qiskit | 0.43.0 |
| qiskit-machine-learning | 0.6.0 |

Tabla 4.2: Tabla de versiones de los diferentes paquetes del software Qiskit.

### Qiskit Runtime

Qiskit Runtime es un servicio de computación cuántica y un modelo de programación que permite agilizar operaciones cuánticas. Su diseño utiliza recursos de computación clásica para ejecutar circuitos en procesadores cuánticos de manera más eficiente.

Estas mejoras de eficiencia son accesibles a través de primitivas. Las primitivas son una interfaz simplificada para definir cargas de trabajo cuántico-clásicas en cuasi tiempo real [17], necesarias para construir y personalizar aplicaciones de manera eficiente. Están diseñadas para ejecutarse en sesiones, que básicamente vinculan el entorno cuántico seleccionado a los trabajos de tu sesión durante un período de tiempo, de manera que no se vean interrumpidos por los trabajos de otros usuarios. Las sesiones se cierran cuando todos los trabajos asociados a ella terminan o cuando esta se cierra o pausa por parte del usuario.

En la figura 4.2 podemos observar como se ejecutan los programas haciendo uso de Qiksit Runtime. Cuando queremos mandar un trabajo a un recurso cuántico tanto desde Qiskit como desde Qiskit Runtime este entra a una cola. Esta cola está diseñada para hacer un reparto equitativo de recursos entre los distintos usuarios que solicitan su uso.

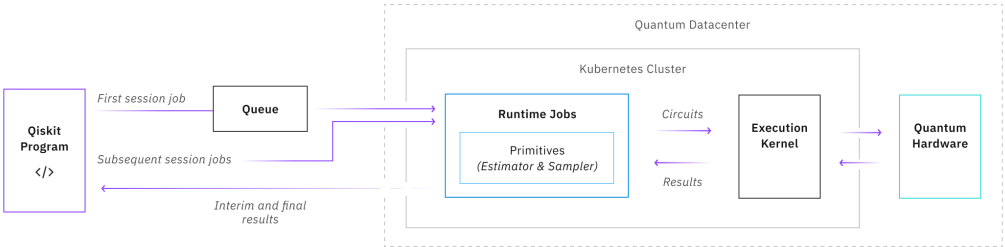


Figura 4.2: Ejemplo de ejecución de un proceso en Qiskit Runtime. Fuente: [2]

El hecho de trabajar con sesiones en vez de con trabajos individuales hace que la ejecución de procesos complejos que implican muchos trabajos sea mucho más eficiente,

ya que el usuario solo tendrá que esperar la cola una vez, en vez de tener que esperar la cola por cada trabajo como sería el caso de las ejecuciones en Qiskit convencional. Este nuevo enfoque ha demostrado acelerar la ejecución de ciertas tareas en un factor x120 si se compara con la ejecución de forma convencional.

Por otro lado, Qiskit Runtime tiene ciertas limitaciones. Al ser una tecnología en una fase muy primaria de su desarrollo, sus funcionalidades se encuentran muy limitadas. Estas funcionalidades están implementadas a través de las primitivas que hemos mencionado anteriormente. Es por esto, que el usuario se encuentra limitado a las primitivas disponibles y a las funcionalidades desarrolladas en base a las mismas. Uno de los retos a los que nos hemos enfrentado en este TFM es el método de ejecución utilizado para evaluar nuestro clasificador.

Cuando iniciamos el desarrollo de este TFM, una de las funcionalidades disponibles en Qiskit Runtime era la implementación del algoritmo de *Quantum Kernel Alignment*, que nos permitía estimar nuestro kernel cuántico y entrenar el clasificador con él.

Por esta razón, y considerando las ventajas que ofrece Qiskit Runtime en términos de tiempos de ejecución, decidimos utilizar este método para estimar nuestro kernel cuántico. Realizamos varias pruebas utilizando Qiskit Runtime en computadoras cuánticas reales y simuladores, obteniendo ejecuciones altamente eficientes gracias a este nuevo enfoque proporcionado por la herramienta.

A pesar de estos resultados iniciales prometedores, una de las desventajas actuales de Qiskit Runtime es su volatilidad. Debido a que se encuentra en una fase muy temprana de desarrollo, es un software en constante cambio y evolución. Esto implica que en diferentes versiones pueden aparecer nuevas primitivas o desaparecer algunas existentes. En nuestro caso, la funcionalidad que utilizábamos para la estimación del kernel fue marcada como obsoleta cuando estábamos en pleno desarrollo de este trabajo. El algoritmo QKA que estábamos utilizando ya no se encontraba disponible en las funcionalidades de Qiskit Runtime. Por esta razón, optamos por utilizar una implementación alternativa del algoritmo en Qiskit para replicar los resultados obtenidos a través de Qiskit Runtime.

Al replicar los resultados, utilizamos simuladores en todos los casos. La razón detrás de esta elección fue que, al no poder utilizar Qiskit Runtime, la ejecución de este proceso tan complejo en computadoras cuánticas reales se volvía inviable debido a los tiempos de espera y al gran número de ejecuciones requeridas.

A pesar de las dificultades enfrentadas, el uso de esta nueva herramienta resultó enriquecedor. No solo fue la primera vez que se utilizó en un TFM, sino que también pudimos comprobar su potencial y las ventajas que este enfoque ofrece en comparación con el enfoque tradicional de Qiskit, convirtiendo a Qiskit Runtime en una tecnología realmente prometedora.

### Recursos Cuánticos

Como hemos mencionado anteriormente en este trabajo, existen dos tipos de recursos cuánticos: ordenadores cuánticos y simuladores

El entorno de computación cuántica *IBM Quantum Experience* cuenta tanto con simuladores como ordenadores cuánticos reales abiertos a cualquier desarrollador de manera

gratuita. Estos recursos gratuitos son limitados y es el motivo por el cual no hemos realizado todos los desarrollos sobre ordenadores cuánticos reales.

Debido al acceso limitado a recursos cuánticos reales y sus largos tiempos de espera, así como la falta de disponibilidad de Qiskit Runtime, nos vimos restringidos en el uso de solo dos ordenadores cuánticos para realizar nuestras ejecuciones. En la tabla 4.3 se presentan los detalles de estos computadores.

| Nombre | Qubits | Procesador | CLOPS | QV |
|---|---|---|---|---|
| *IBMQ Montreal* | 27 | Falcon r5.11 | 2K | 128 |
| *IBMQ Nairobi* | 7 | Falcon r5.11H | 2.6K | 32 |

Tabla 4.3: Tabla de ordenadores cuánticos utilizados para ejecuciones

Estos ordenadores cuánticos se utilizaron como base para evaluar la viabilidad del uso de simuladores. Estos simuladores son accesibles a través de Qiskit Aer, un simulador de alto rendimiento que permite un modelo de ruido altamente configurable. Su núcleo está escrito en C++ para mayor velocidad.

El simulador *IBMQ Statevector Simulator* es el simulador principal de Qiskit Aer. *IBMQ Statevector Simulator* simula la ejecución de circuitos cuánticos en un dispositivo real y devuelve medidas de los disparos emitidos. Para evaluar su desempeño hemos ejecutado de manera idéntica en un ordenador cuántico real y en este simulador dos casos diferentes:

1. El problema de 7 qubits generado en este TFM. El ordenador cuántico elegido será *IBMQ Nairobi* ya que es el procesador de referencia utilizado.

2. El problema de 7 qubits generado por IBM. En este caso el ordenador cuántico elegido será *IBMQ Montreal* ya que en este caso se utilizó un sub-grafo de dicho procesador para la generación de los datos.

Los resultados obtenidos en el recurso cuántico real y el simulador fueron idénticos. Este hecho es un gran indicador de la viabilidad de este tipo de tecnología para replicar los resultados que obtendríamos en un ordenador cuántico real.

Es por esto por lo que de ahora en adelante utilizaremos *IBMQ Statevector Simulator* como nuesto recurso cuántico para el resto de ejecuciones. Esto facilitará mucho los desarrollos ya que reduce de manera significativa los tiempos de ejecucción.

Por otro lado, para la generación de datos nos hemos seguido basando en arquitecturas de ordenadores cuánticos reales. En la Tabla 4.4 podemos ver los detalles de dichos sistemas.

| Nombre | Qubits | Procesador | CLOPS | QV |
|---|---|---|---|---|
| *IBMQ Belem* | 5 | Falcon r4T | 2.5K | 16 |
| *IBMQ Manila* | 5 | Falcon r5.11L | 2.8K | 32 |
| *IBMQ Nairobi* | 7 | Falcon r5.11H | 2.6K | 32 |
| *IBMQ Guadalupe* | 16 | Falcon r4P | 2.4K | 32 |
| *IBMQ Kolkata* | 27 | Falcon r5.11 | 2K | 128 |

Tabla 4.4: Tabla de ordenadores cuánticos utilizados para generación de datos

### 4.2.3. Entorno de Ejecución

Para realizar los desarrollos necesarios para llevar a cabo el estudio de los clasificadores cuánticos descritos en el capítulo anterior, utilizaremos Notebooks de Python.

Hemos elegido esta tecnología debido a su gran ecosistema de librerías para manipulación y carga de datos, visualización y *Machine Learning* así como la posibilidad de trabajar con los paquetes desarrollados por Qiskit para Python.

Estos Notebooks los ejecutaremos en entornos en la nube de *IBM Quantum Lab*. En la tabla 4.5 podemos ver la configuración de estos entornos de ejecución.

| Información del Sistema | |
|---|---|
| Python version | 3.10.8 |
| Python compiler | GCC 10.4.0 |
| OS | Linux |
| CPUs | 1 |
| Cores | 8 |
| Memoria[Gb] | 32 |

Tabla 4.5: Tabla de información del entorno de ejecución de Python.

## 4.3. Códigos Desarrollados

En esta sección detallaremos los distintos códigos que se han utilizado para el desarrollo de las pruebas discutidas en este TFM. Todos los códdigos desarrollados se pueden encontrar en el Github de este TFM en el siguiente **link** [18].

Los códigos desarrollados se podrían dividir en dos:

1. **Generación de datos**: Este código se utilizará para la generación de los distintos datasets utilizados en las pruebas. Para su creación nos hemos basado en el punto **3.4**. Con estas premisas se han generado datos para 5, 7, 10 y 16 qubits.

2. **Evaluación del clasificador**: Este código se utilizará para evaluar el clasificador cuántico sobre los datos generados. Su creación está basada en el siguiente ejemplo de *IBM Quantum* [19].

En nuestro caso no solo evaluaremos el clasificador cuántico, si no que también evaluaremos los resultados obtenidos por una versión clásica de nuestro clasificador (SVM).

### 4.3.1. Generación de Datos

En esta sección detallaremos la estructura principal los programas desarrollados en MATLAB para la generación de datos. La estructura de este programa viene determinada por el proceso descrito en el punto **3.4**.

El desarrollo de este proceso ha sido uno de los puntos más complejos de este TFM. La generación de datos es un paso crítico en la investigación, ya que los datos utilizados para entrenar y evaluar el clasificador tienen un impacto directo en los resultados y conclusiones del estudio.

Se ha prestado especial atención a este proceso, con el objetivo no solo de obtener conjuntos de datos representativos y relevantes para el problema abordado si no también de desarrollar una metodología que permita generar datos para cualquier arquitectura.

Durante el desarrollo de este proceso, se han enfrentado desafíos técnicos, como por ejemplo la necesidad de utilizar un servidor dedicado en ocasiones, y se han realizado iteraciones y ajustes constantes. Se ha trabajado en colaboración con expertos en el campo y se han llevado a cabo experimentos rigurosos para validar la calidad y la adecuación de los conjuntos de datos generados.

Detallaremos ahora dicho proceso utilizando como ejemplo el caso de 16 qubits.

1. **Definición de parámetros iniciales**: Lo primero que haremos en nuestro programa es definir los distintos parámetros que utilizaremos en la generación de datos.
   - Número de muestras
   - Varianza del error aleatorio aplicado
   - Semilla
   - Set de ángulos aleatorios $\theta_{\pm} \in [-\pi/2, \pi/2]^{2n}$ con $n$ el número de qubits de nuestro caso

   Además, también definiremos la matrices de Pauli que utilizaremos en los cálculos posteriores.

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

$$\sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$

$$\sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

2. **Definición del grafo de conectivdiad**: Seleccionaremos el procesador que utilizaremos como referencia para generar los datos. Si en el caso que vamos a abordar, no utilizamos todos los qubits del procesador, seleccionaremos el subgrafo que utilizaremos.

En nuestro caso hemos utilizado todos los qubits del procesador del computador cuántico *IBMQ Guadalupe*. En la figura 4.3 podemos ver la arquitectura de dicho recurso cuántico.

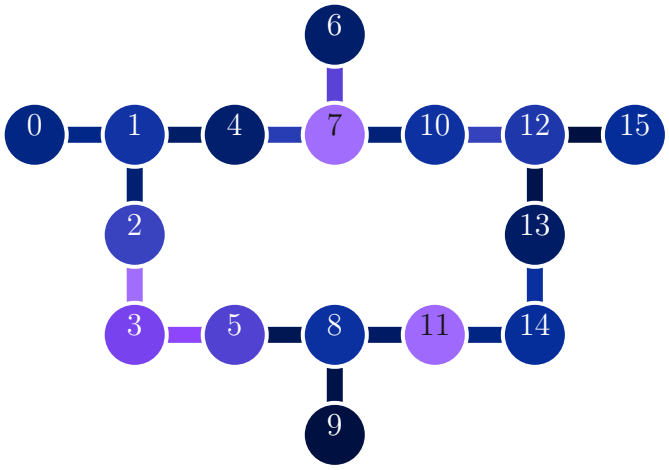


Figura 4.3: Grafo de conectividad del procesador de *IBMQ Guadalupe*

Podemos observar que dicho procesador tiene 16 conexiones entre qubits. Estas conexiones son las que debemos representar a través del estabilizador del grafo $S_{graph}$.

$$S_{graph} = \langle \{X_i \bigotimes_{k:(k,i)\in E} Z_k\}_{i\in V} \rangle \tag{4.1}$$

3. **Definición de rotaciones**: Debemos definir las rotaciones $R(\theta)$ con $\theta \in \theta_{\pm}$.

$$R(\theta) = R(\theta_1, \theta_2) \otimes R(\theta_3, \theta_4) \otimes ... \otimes R(\theta_{31}, \theta_{32}) \tag{4.2}$$

Con $R(\theta_i, \theta_j) = exp(-i(\theta_i/2)X)exp(-i(\theta_j/2)Z)$.

Esta definición de la matriz $R$ nos servirá para definir los distintos cosets $C_{\pm}$ de la forma $C_{\pm} = RX_iZ_j$.

En el siguiente paso definiremos la matriz de interacción $X_iZ_j$.

4. **Definición matricial de las conexiones**: Una vez tenemos nuestro grafo de conetividad y nuestro operador $S_{graph}$, obtendremos la representación matricial de cada conexión. En este caso vamos a ejemplificar este proceso con los qubits 0 y 1.

   Haciendo uso de 4.1 podemos definir la primera conexión de la siguiente forma:

$$X_0Z_1 = X_0 \otimes Z_1 \otimes I_3 \otimes ... \otimes I_{16} \tag{4.3}$$

   Donde $I$ representa la matriz identidad y los subíndices el qubit asociado a dicha matriz.

5. **Creación del coset**: Crearemos la matriz del coset para la interacción entre los qubits 0 y 1.

$$C(0,1) = RX_0Z_1 \tag{4.4}$$

6. **Estimacion de los ángulos de Euler de los qubits de interacción**: En este punto estimaremos los nuevos ángulos de Euler con los cuales podemos representar los qubits de interacción. Esto se debe a que los únicos ángulos de Euler afectados cuando creamos el coset son los asociados a la conexión que estamos estudiando, en nuestro caso los dos primeros qubits.

   Para hacer esto lo primero que haremos será generar una versión reducida de nuestro coset utilizando únicamente los qubits de interacción. Esta versión reducida será de la forma:

$$R(\theta_{01}) = R(\theta_1, \theta_2) \otimes R(\theta_3, \theta_4) \tag{4.5}$$

$$X_0 Z_1 = X_0 \otimes Z_1 \tag{4.6}$$

$$C_{01} = R(\theta_{01}) X_0 Z_1 = R(\phi_1, \phi_2) \tag{4.7}$$

   Una vez tenemos la versión reducida de nuestro coset, utilizaremos (3.9) para obtener los ángulos $(\phi_1, \phi_2)$ asociados al operador $C_{01}$ que hemos obtenido.

   Con estos ángulos generaremos un nuevo coset:

$$C' = R(\phi_1, \phi_2) \otimes R(\theta_3, \theta_4) \otimes ... \otimes R(\theta_{31}, \theta_{32}) \tag{4.8}$$

   Si la estimación de ángulos realizada en este paso ha sido correcta, la nueva matriz $C'$ debería ser igual al coset original.

7. **Comparación del coset original con el generado tras la estimación de ángulos**: Como hemos mencionado en el paso anterior, si la estimación de ángulos realizada es correcta, las matrices $C$ y $C'$ deberían ser equivalentes.

   Consideraremos que la estimación de los nuevos ángulos de Euler ha sido exitosa si se cumple la siguiente condición:

$$\|C - C'\| < 0{,}001 \tag{4.9}$$

   En dicho caso, generaremos una observación en nustro set de datos. Esta observación estará compuesta por los ángulos de Euler asociados al coset obtenido y un indicador que valdrá $+1$ si el coset obtenido era $C_+$ y $-1$ para el caso de $C_-$.

8. **Cálculo de los datos**: Para cada una de las distintas conexiones existentes replicaremos los cálculos descritos en los puntos del 4 al 7. Además, este proceso se ejecutará para $C_+$ y para $C_-$.

   Como resultado de esto, obtendremos los ángulos de Euler de los diferentes cosets así como el indicador de a que coset pertenece cada observación. El número de observaciones generadas en este proceso dependerá del número de conexiones de nuestro grafo de conectividad, ya que por cada conexión obtendremos una observación para $C_-$ y otra para $C_+$.

9. **Aplicación de ruido**: Una vez tenemos definidos nuestros cosets, aplicaremos a cada observación ruido gausiano. Este ruido está controlado por el parámetro de varianza definido a principio del programa.

   Estas nuevas observaciones se generan aplicando la siguiente fórmula a todos los ángulos de cada observación.

   $$\phi' = \phi(1 + \epsilon) \tag{4.10}$$

   Donde $\epsilon$ es el término de ruido obtenido de una distribución gausiana con media cero y varianza igual a la definida en los parámetros iniciales del programa.

Con esta estructura, generaremos datos para 5,7,10 y 16 qubits. En la siguiente tabla podemos observar los detalles de cada uno de estos códigos.

| Programa | Qubits | Procesador Referencia |
|---|---|---|
| Five_qbit_sample_generation_v1.m | 5 | *IBMQ Belem* |
| Five_qbit_sample_generation_v2.m | 5 | *IBMQ Manila* |
| Seven_qbit_sample_generation_v0.m | 7 | *IBMQ Nairobi* |
| Ten_qbit_sample_generation_v0.m | 10 | *IBMQ Kolkata* |
| Sixteen_qbit_sample_generation_v0.m | 16 | *IBMQ Guadalupe* |

Tabla 4.6: Tabla de códigos para generación de datos

En el caso del código que genera los datos para el caso de 16 qubits, la aplicación de ruido se realiza a través de un código Python independiente. Esto se debe a que la complejidad de los cálculos necesarios para la generación de este conjunto de datos nos obligó a utilizar un servidor dedicado, ya que un ordenador convencional no contaba con suficientes recursos. En este servidor se ejecutaron los ocho primeros puntos definidos en esta sección.

El código desarrollado para la aplicación de ruido es *Add_Noise_16qb.ipynb*.

### 4.3.2. Evaluación del Clasificador

Una vez generados los datos, nuestro objetivo será evaluar el desempeño del clasificador cuántico en sobre dichos datos. Además, compararemos estos resultados con los obtenidos por un clasificador clásico.

Para evaluar y comparar estos clasificadores lo primero que haremos será crear una muestra de entrenamiento, sobre la cual entrenaremos los clasificadores, y una muestra de test para poder evaluar su desempeño sobre ella. Estas muestras se obtendrán con sampleo aleatorio sin reemplazamiento del set de datos original.

La métrica utilizada para evaluar el desempeño de los clasificadores será la precisión en test.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.11}$$

Donde $TP + TN$ representa el número de aciertos y $TP + TN + FP + FN$ el número total de observaciones evaluadas.

El Notebook base que utilizaremos será el mismo en todas las pruebas, modificando únicamente los datos de entrada y el grafo de conexiones del mapa de características en función del caso a resolver. Podemos dividir el Notebook desarrollado en las siguientes secciones:

1. **Configuración del entorno de ejecución**: En esta primera sección configuraremos el entorno de ejecución. En este punto comprobaremos que existen los paquetes necesarios en nuestro sistema y si no fuera así los descagaríamos.

2. **Importación de librerías**: Una vez contamos con todos los recursos necesarios accesibles en nuestro entorno, procederemos a importar las librerías necesarias para el estudio de nuestro clasificador. En la tabla 4.7 podemos ver las principales librerías utilizadas.

| Librerías |
|---|
| OS |
| Sys |
| IO |
| Time |
| Pandas |
| Numpy |
| Sklearn |
| Matplotlib |
| Qiskit |

Tabla 4.7: Tabla de librerías utilizadas

   Además de estas librerías principales, se han utilizado funciones desarrolladas por *IBM Quantum* en el repositorio **prototype-quantum-kernel-training**

3. **Carga y preparación de datos**: Lo siguiente que haremos será cargar los datos del problema que podamos estudiar como un *DataFrame* de Pandas.

   Posteriormente, dividiremos dichos datos en entrenamiento y test mediante sampleo aleatorio sin remplazamiento.

4. **Definición del mapa de características**: Para realizar el estudio de nuestro problema, necesitamos definir el mapa de características que vamos a utilizar para el cálculo del kernel. Este mapa de características está determinado por la arquitectura del procesador que hemos utilizado para el proceso de generación de datos y por ello deberá ser modificado antes de la ejecución de las distintas pruebas.

5. **Configuración del clasificador**: Una vez definido el mapa de características, definiremos todo lo necesario para ejecutar nuestro clasificador cuántico.

   - Entorno de Ejecución: Indicaremos el computador cuántico o simulador a utilizar para el cálculo del kernel.

- Optimizador: Configuraremos el optimizador SPSA (iteraciones máximas, learning rate y perturbación)
- Estimador del Kernel: Podemos modificar el punto inicial de los parámetros de nuestro circuito. Por defecto utilizaremos 0.1 como punto inicial ya que es el valor utilizado en [19].

6. **Estimación del kernel**: Con todo lo necesario configurado, podemos estimar el kernel que más tarde utilizaremos en nuestra SVM. Este cálculo se ejecuta en el entorno cuántico seleccionado.

7. **Entrenamiento del clasificador cuántico**: Con la matriz del kernel ya calculada, entrenaremos nuestra SVM sobre los datos de entrenamiento y evaluaremos su rendimiento sobre los datos de test.

8. **Visualización de resultados**: En esta sección graficaremos la función de pérdida como función de los pasos del optimizador así como un mapa de calor de la matriz del kernel. Esto nos ayudará a interpretar mejor los resultados obtenidos.

   En el gráfico de pérdida buscamos un descenso gradual de la función de pérdida hasta un punto en el que esta no descienda más, esto significa que la función de pérdida ha alcanzado un mínimo y por lo tanto nuestro clasificador no aprendería más con más iteraciones.

   Para el caso del mapa de calor de la matriz del kernel, lo que esperamos observar es una estructura diagonal, debido a que nuestro kernel mapea los datos de entrada a dos estados principales correspondientes a los dos cosets que estamos intentando identificar. Debido al ruido que hemos añadido estos estados principales se pueden ver ligeramente alterados pero este hecho no debería impedir al clasificador identificarlos de manera correcta.

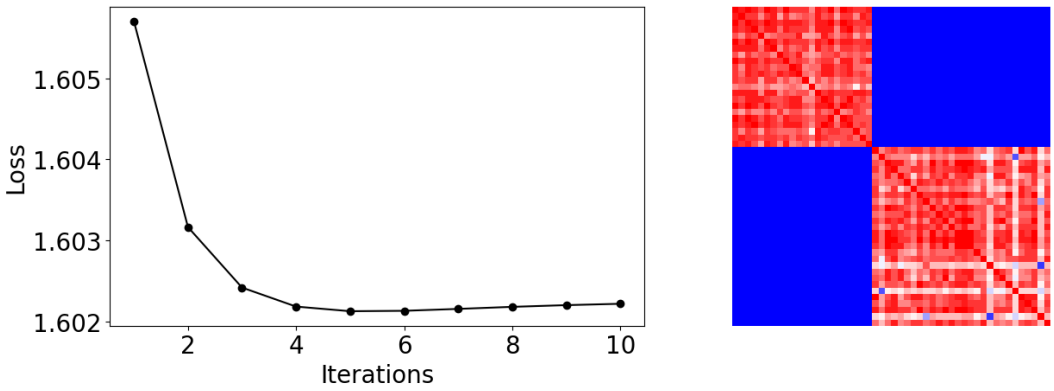


Figura 4.4: Ejemplo de gráficos resultantes del entrenamiento del clasificador cuántico

9. **Entrenamiento del clasificador clásico**: Por último, configuraremos un clasificador clásico y lo evaluaremos de igual forma.

   Utilizaremos como clasificador una SVM pero en este caso todos sus componentes serán clásicos.

   Para la selección de los distintos parámetros de esta SVM hemos hecho uso de una búsqueda en rejilla con validación cruzada. Esto nos permitirá evaluar un gran número de combinaciones y seleccionar aquella que mejor desempeño proporcione.

Esta búsqueda en rejilla contiene los siguientes parámetros:

- C: Es un parámetro de regularización que nos permitirá reducir el overfitting de nuestro clasificador. Actua como una penalización L2.
- Gamma: Coeficiente del kernel. Un valor muy alto de gamma podría llevar a overfitting, mientras que valores demasiado bajos podrían hacer que nuestra máquina no aprendiese lo suficiente.
- Kernel: Especifica el tipo de kernel a utilizar. En nuestro caso usaremos el kernel polinómico y el kernel radial.
- Degree: Grado del kernel polinómico. Para el caso del kernel radial este parámetro se ignora.
- Coef0: Término independiente de la función del kernel. Este parámetro se utiliza únicamente en el kernel polinómico.

Una vez escogido el mejor clasificador, evaluaremos su presisión sobre los datos de test, para poder comparar estos resultados con los obtenidos por el clasificador cuántico.

Siguiendo este esquema se han realizado las siguientes pruebas:

| Programa | Qubits | Sistema Cuántico | Tecnología |
|---|---|---|---|
| qkt_5qb_SV.ipynb | 5 | *IBMQ Statevector* | Qiskit QKT |
| qkt_5qb-Manila_SV.ipynb | 5 | *IBMQ Statevector* | Qiskit QKT |
| qkt_7qb_SV.ipynb | 7 | *IBMQ Statevector* | Qiskit QKT |
| qka_exec_Nairobi-7Qubit_Matlab.ipynb | 7 | *IBMQ Nairobi* | Qiskit Runtime |
| qka_exec_Montreal-7_Qubit$_IBM.ipynb$ | 7 | *IBMQ Montreal* | Qiskit Runtime |
| qkt_10qb_SV.ipynb | 10 | *IBMQ Statevector* | Qiskit QKT |
| qkt_16qb_SV.ipynb | 16 | *IBMQ Statevector* | Qiskit QKT |

Tabla 4.8: Tabla de ejecuciones

## 4.4. Conclusiones

En este capitulo se han detallado los desarrollos realizados en este TFM. Se han descrito las tecnologías utilizadas para el desarrollo de los distintos puntos del estudio, así como los retos técnicos a los que nos hemos enfrentado como la necesidad de utilizar un servidor dedicado o el hecho de que Qiskit Runtime dejase de ser una opción de ejecución válida en plena fase de desarrollo de este TFM.

También se ha presentado de manera detallada la metodología desarrollada para la generación de nuevos sets de datos dada una arquitectura de procesador. Como hemos mencionado en este capítulo este desarrollo ha sido uno de los más complejos de nuestro trabajo.

Por último, se han descrito los diferentes puntos que componen el Notebook de Python utilizado para la evaluación del clasificador.

En el siguiente capítulo discutiremos los resultados obtenidos en cada una de las pruebas realizadas.

# 5 Resultados

## 5.1. Introducción

En este capítulo discutiremos los resultados obtenidos en las distintas pruebas realizadas y descritas en 4.

Como hemos mencionado anteriormente, la mayoría de desarrollos se han ejecutado sobre simuladores cuánticos. En los casos en los que también se ha utilizado un sistema cuántico real los resultados obtenidos han sido idénticos a los conseguidos con el simulador.

Este hecho es un muy buen indicador de que el algoritmo utilizado es robusto ante el ruido que puede existir en un ordenador cuántico real, además de confirmar la viabilidad del uso de simuladores para replicar los resultados que obtendríamos en un ordenador cuántico real cuando este no esté disponible.

Es por esto que en esta sección únicamente discutiremos las pruebas ejecutadas en simuladores cuánticos.

## 5.2. Pruebas 5 qubits

Como hemos detallado en 4 se han desarrollado 2 datasets para el problema de 5 qubits. El proceso que nos llevó a realizar los desarrollos para dos arquitecturas distintas es el siguiente.

En una primera instancia, elegimos *IBMQ Belem* como el ordenador cuántico de referencia para este problema. Este ordenador cuenta con un procesador en forma de T como el que podemos observar en la figura 5.1.

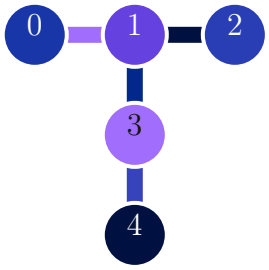


Figura 5.1: Grafo de conectividad de *IBMQ Belem*.

Observamos que el clasificador, a la hora de optimizar la función de coste, esta se reducía mucho en la primera iteración pero a partir de ahí no se conseguía reducir más, llegando incluso a empeorar con el paso de las iteraciones. Pese a esto, el clasificador proporcionaba un 100 % de precisión entrenamiento y test.

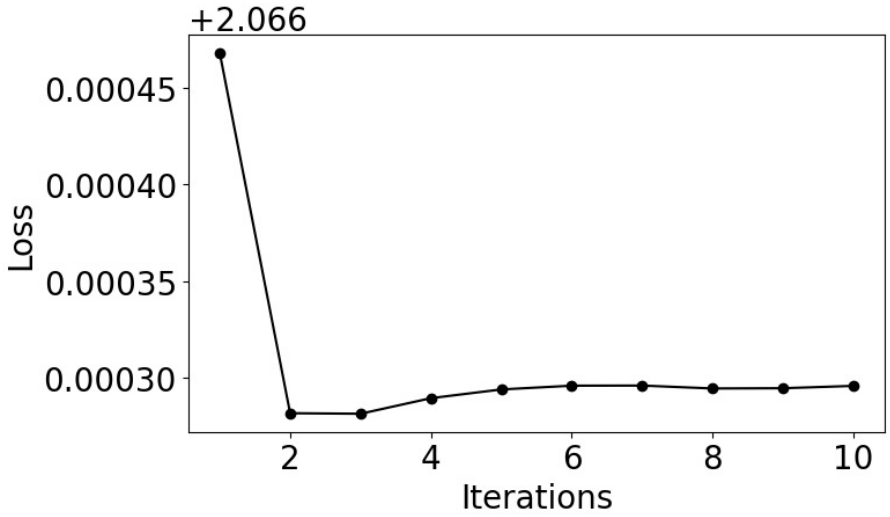


Figura 5.2: Iteraciones del optimizador de la función de coste para los datos generados con la arquitectura de *IBMQ Belem*.

Además, si observamos el mapa de calor del kernel, podemos observar como principalmente existen dos estados muy bien diferenciados, rojo y azul, con muy pocos puntos de colores intermedios que representarían estados intermedios entre cada uno de los cosets.

Este hecho nos hizo pensar que el problema que estábamos generando con la arquitectura elegida en un principio, la del procesador de *IBMQ Belem*, era un problema trivial

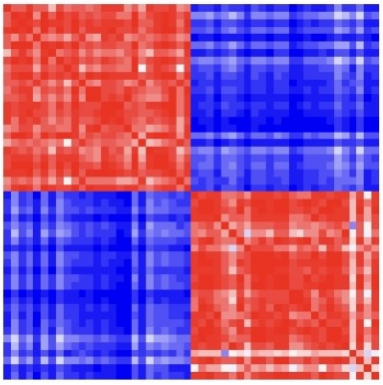

Figura 5.3: Mapa de calor del kernel estimado para los datos generados con la arquitectura de *IBMQ Belem*.

que apenas necesitaba del optimizador de la SVM para encontrar una frontera adecuada capaz de distinguir ambos cosets perfectamente.

Debido a la existencia de ordenadores cuánticos de 5 qubits con distinta arquitectura, decidimos evaluar si este hecho era debido a la arquitectura seleccionada.

Procedimos a generar un nuevo problema de 5 qubits en este caso con la arquitectura de *IBMQ Manila*. Esta arquitectura es una arquitectura lineal como la que vemos en la figura 5.4.


Figura 5.4: Grafo de conectividad de *IBMQ Manila*.

Los resultados obtenidos con este nuevo set de datos son también del 100 % tanto en entrenamiento como en test, pero en este caso nuestra función de coste si que vemos que se optimiza con cada iteración. Por otro lado el mapa de calor del kernel tambien presenta la forma que cabría esperar, con la mayoría de los puntos en dos estados muy diferenciados y ciertos puntos en estados intermedios.

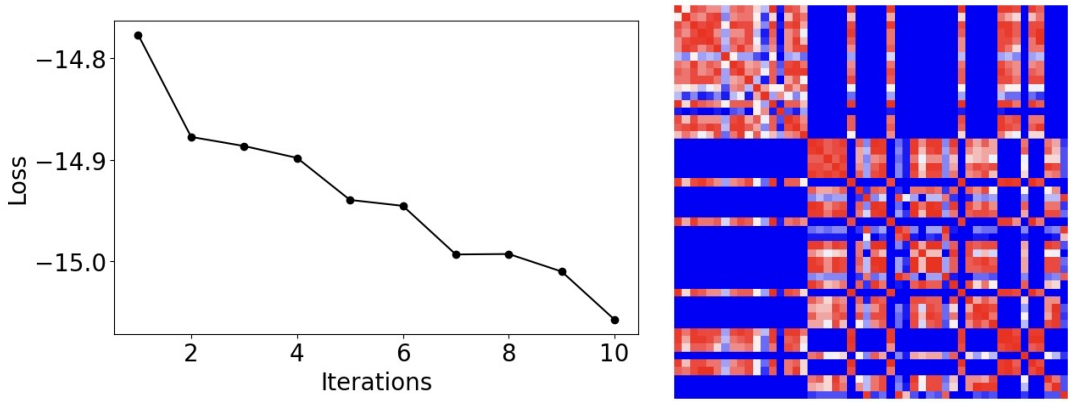


Figura 5.5: a) Iteraciones del optimizador de la función de coste para los datos generados con la arquitectura de *IBMQ Manila*. b) Mapa de calor del kernel estimado para los datos generados con la arquitectura de *IBMQ Manila*.

Para ambas arquitecturas discutidas, los resultados obtenidos por el clasificador clásico son también del 100 % en test.

Estos resultados nos indican que dependiendo de la arquitectura del procesador que tomemos como referencia, el problema que generamos puede ser más complejo de solucionar por nuestro clasificador. Además, podemos observar que tanto el clasificador cuántico como el clásico proporcionan buenos resultados en este primer caso.

Dados estos buenos resultados, iremos añadiendo complejidad a nuestro problema añadiendo qubits de manera progresiva con el objetivo de encontrar un punto en el que nuestro clasificador cuántico supere a su contraparte clásica.

## 5.3. Pruebas 7 qubits

El siguiente problema que intentaremos resolver es el caso de 7 qubits. En la figura 5.6 podemos observar el grafo de conectividad de *IBMQ Nairobi*.

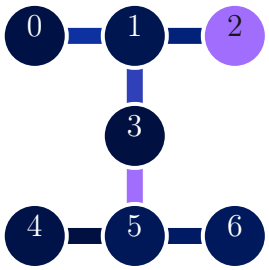


Figura 5.6: Grafo de conectividad de *IBMQ Nairobi*.

Actualizaremos la definición de nuestro kernel en base a dicho grafo de conectividad y ejecutaremos de nuevo el proceso.

Podemos observar en la figura 5.7 los gráficos resultado del proceso de entrenamiento.

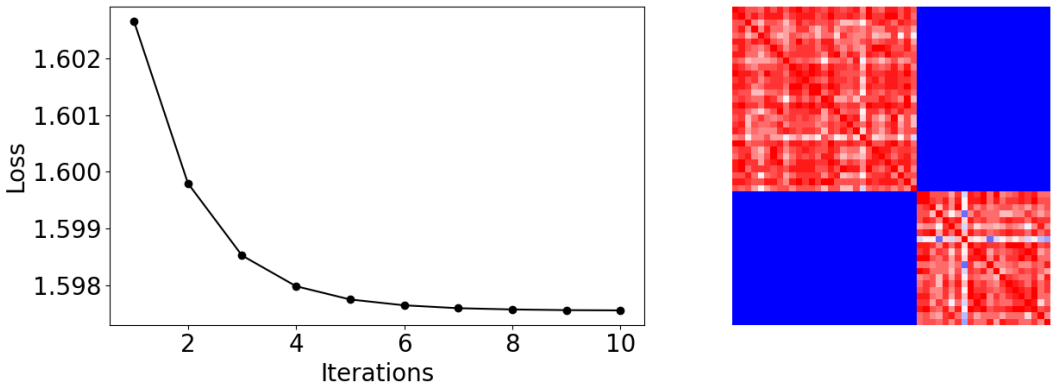


Figura 5.7: a) Iteraciones del optimizador de la función de coste para los datos generados con la arquitectura de *IBMQ Nairobi*. b) Mapa de calor del kernel estimado para los datos generados con la arquitectura de *IBMQ Nairobi*.

Como cabría esperar, el optimizador minimiza el valor de la función de coste hasta acanzar un valor estable entorno a la octava iteración. Esto es un buen indicador de que

el proceso de entrenamiento ha sido correcto donde la máquina de vectores de soporte ha alcanzado un mínimo de la función de coste.

Si nos fijamos en el mapa de calor del kernel también podemos apreciar que apenas existen estados intermedios, caracterizándose la mayoría de puntos por el estado azul o por el rojo, donde cada uno de estos colores se corresponde con uno de los cosets definidos.

La precisión obtenida con este entrenamiento es de un 100 % en test. En el caso del entrenamiento del clasificador clásico, el resultado obtenido es el mismo.

Una vez más hemos obtenido muy buenos resultados tanto clásica como cuánticamente.

## 5.4. Pruebas 10 qubits

El siguiente nivel de complejidad desarrollado es el caso de 10 qubits. Para este caso escogimos como procesador de referencia el procesador de *IBMQ Kolkata*. Este computador cuántico tiene 26 qubits y una arquitectura de doble anillo como la que podmeos observar en la figura 5.8.

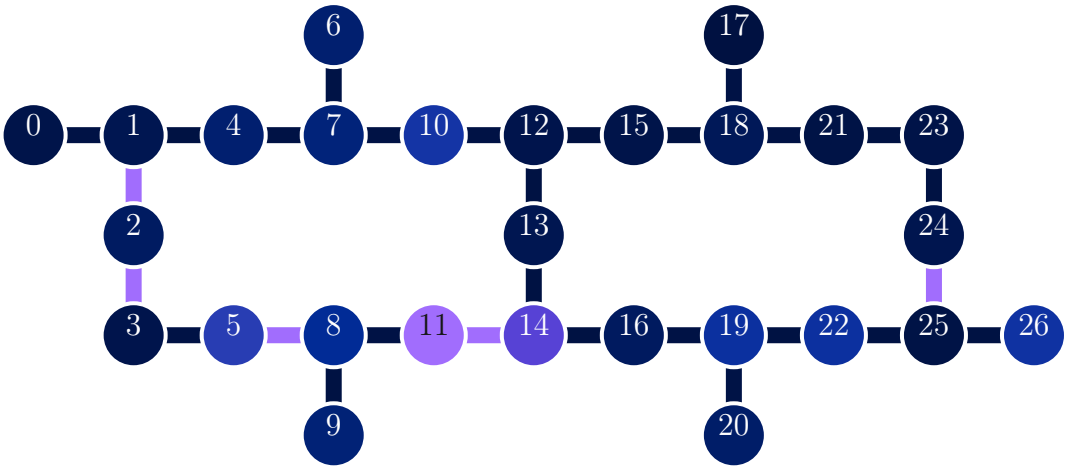


Figura 5.8: Grafo de conectividad de *IBMQ Kolkata*.

Como este procesador contiene más qubits que los que necesitamos, seleccionaremos un subgrafo para utilizarlo en nuestro problema. Este subgrafo es necesario tanto en la generación de los datos como en la definición del mapa de caractrísticas.

En la figura 5.9 podemos ver señalados los qubits utilizados para la resolución de este problema. El subgrafo seleccionado es un tramo lineal del procesador con una arquitectura muy similar a la utilizada en el caso de 5 qubits en *IBMQ Manila*.

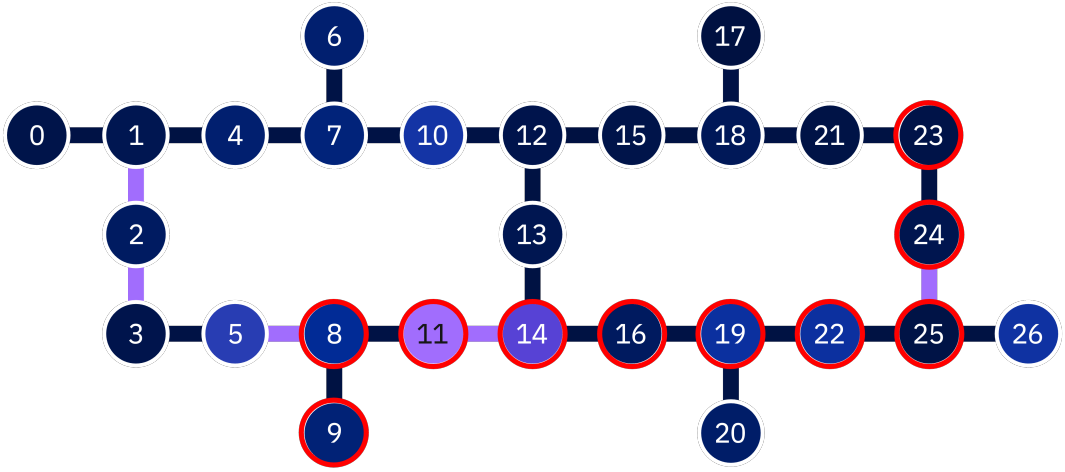


Figura 5.9: Qubits utilizados del ordenador *IBMQ Kolkata*

Una vez seleccionados los qubits que vamos a utilizar podemos redefinir nuestro mapa de características en función de sus conexiones y entrenar nuestro clasificador.

En la figura 5.10 tenemos los resultados del entrenamiento de nuestro clasificador cuántico. Como en casos anteriores, podemos observar que el optimizador a funcionado correctamente alcanzando un mínimo como esperaríamos.

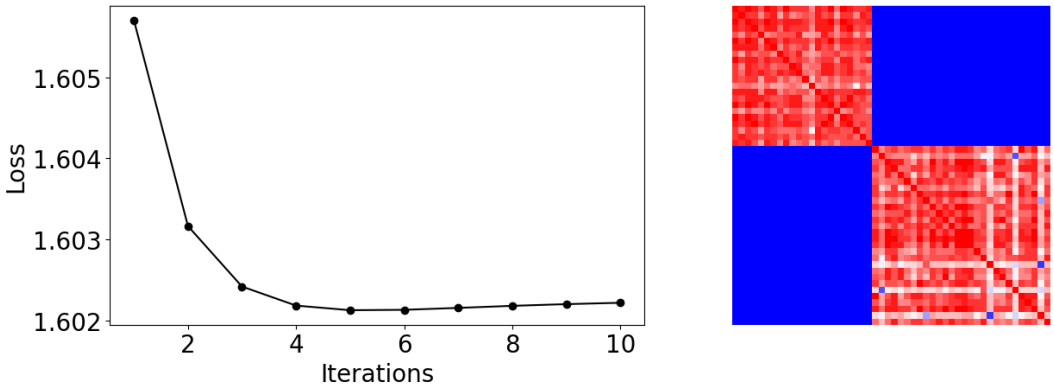


Figura 5.10: a) Iteraciones del optimizador de la función de coste para los datos generados con la arquitectura de *IBMQ Kolkata*. b) Mapa de calor del kernel estimado para los datos generados con la arquitectura de *IBMQ Kolkata*.

Tambien podemos observar como las dos clases están bien diferenciadas en el mapa de calor del kernel.

Si evaluamos ahora las métricas del clasificador cuántico, obtenemos un 100 % de precisión en test. En el caso del clasificador clásico, los resultados en test son de un 93,3 %. Este es el primer caso en el que hemos conseguido una ventaja cuántica a la hora de resolver el problema que estamos estudiando.

Con estos resultados, generaremos un nuevo set de datos más complejo. El objetivo será obtener de nuevo una ventaja cuántica de nuestro clasificador sobre su contraparte clásica.

## 5.5. Pruebas 16 qubits

El máximo nivel de complejidad desarrollado es el caso de 16 qubits.Para este caso escogimos como procesador de referencia el procesador de *IBMQ Guadalupe*. Este computador cuántico tiene 16 qubits y una arquitectura de anillo como la que podmeos observar en la figura 5.11.

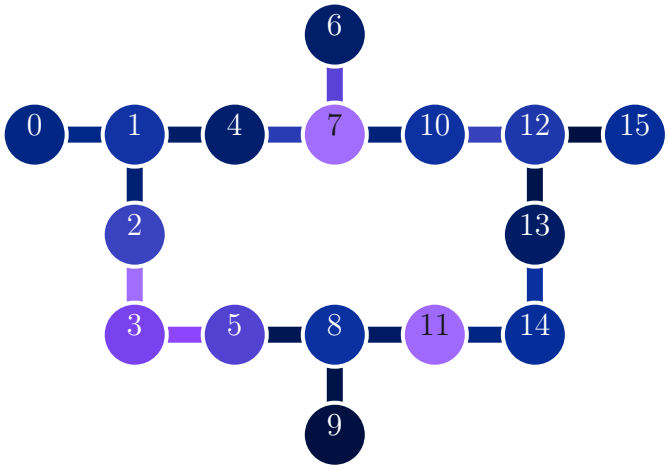


Figura 5.11: Grafo de conectividad de *IBMQ Guadalupe*.

Con este grafo de conectividad configuraremos nuestro mapa de características y entrenaremos el clasificador cuántico.

Podemos observar en la figura 5.12 que el gráfico de optimización de la función de coste no presenta un descenso tan gradual como el observado en otros casos, ya que a partir de la 4 iteración vemos como el optimizador encuentra un mínimo. En el caso del mapa de calor del kernel, predomina la presencia del estado azul sobre el rojo, hecho que en ningún otro caso había sucedido, siendo la distribución de clases del 50 % en todos los casos.

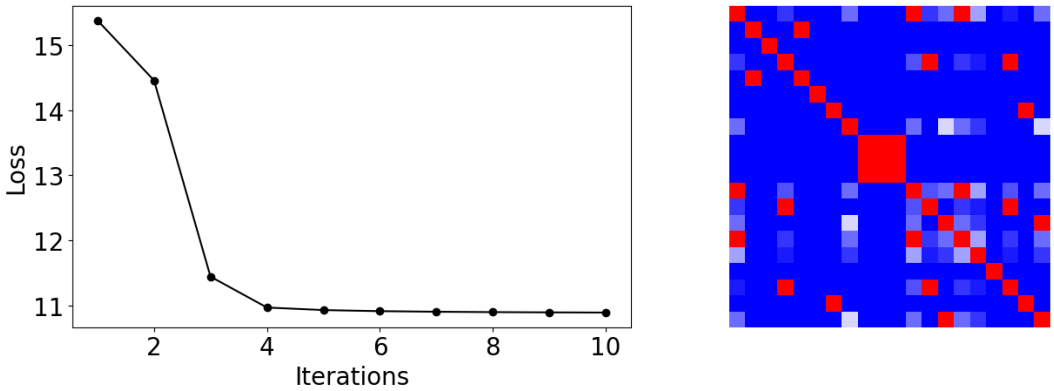


Figura 5.12: a) Iteraciones del optimizador de la función de coste para los datos generados con la arquitectura de *IBMQ Guadalupe*. b) Mapa de calor del kernel estimado para los datos generados con la arquitectura de *IBMQ Guadalupe*.

Estos hechos podrían ser indicadores de que nuestro clasificador cuántico no está siendo capaz de distinguir correctamente ambas clases como sí había conseguido en los

casos anteriores. Si evaluamos la precisión de nuestro clasificador sobre los datos de test, obtenemos un 62,5 %, una cifra menor que la conseguida en las otras pruebas ejecutadas.

Pese a este hecho, el objetivo de la prueba es comprobar si el clasificador cuántico ofrece algún tipo de ventaja frente al clásico. En esta prueaba el clasificador clásico obtiene un 55 % de precisión sobre los datos de test.

Estos resultados, pese a ser más bajos que en el resto de pruebas, sí que resultan relevantes ya que al igual que en el caso anterior, el clasificador cuántico clasifica mejor los datos que su contraparte clásica.

## 5.6. Conclusiones

En este capítulo hemos discutido los resultados obtenidos en las distintas pruebas. El primer resultado destacable es el hecho de que todas las pruebas que han sido ejecutadas tanto en simulador como en ordenador cuántico, han tenido el mismo resultado.

Este hecho postula los simuladores cuánticos como una muy buena alternativa a los ordenadores cuánticos cuando estos no son accesibles.

Por otro lado, en todas las pruebas realizadas nuestro clasificador cuántico ha sido mejor o igual que su contraparte clásica. En los problemas más sencillos de 5 y 7 qubits ambos clasificadores conseguían un 100 % de precisión pero en problemas más complejos hemos observado como nuestro clasificador cuántico superaba al clásico. En la tabla 5.1 podemos ver un resumen de estos resultados.

| Prueba | Clasificador Cuántico | Clasificador Clásico |
|---|---|---|
| 5 qubits Belem | 100 % | 100 % |
| 5 qubits Manila | 100 % | 100 % |
| 7 qubits | 100 % | 100 % |
| 10 qubits | 100 % | 93,3 % |
| 16 qubits | 62,5 % | 55 % |

Tabla 5.1: Resumen de resultados de las distintas pruebas

Estos resultados son muy buenos indicadores de que los kernels cuánticos son capaces de ofrecer ventajas sobre los clásicos para ciertos problemas.

# 6
# Conclusiones y Trabajo Futuro

## 6.1. Introducción

En este capitulo detallaremos las conclusiones obtenidas de todo el trabajo que se ha desarrollado en este TFM. Propondremos también las vías de trabajo futuro que podrían enriquecer este primer estudio.

## 6.2. Conclusiones

La motivación de este TFM era probar que existen ciertos problemas de clasificación que podemos resolver de manera más precisa mediante el uso de kernels cuánticos en vez de únicamente utilizar kernels clásicos.

En [1] encontramos un ejemplo que demuestra dicha suposición para un problema de 26 qubits. Tomando dicho paper como base, hemos diseñado un flujo de trabajo que hace posible la creación de distintos problemas en base a una arquitectura dada. Con este flujo de trabajo, decidimos resolver este problema para 5, 7, 10 y 16 qubits con el objetivo de comprobar si el resultado obtenido en el caso de 26 qubits seguía siendo cierto en estos casos más sencillos.

En base al paper citado generamos los datos necesarios para evaluar los casos de 5, 7, 10 y 16 qubits y configuramos el clasificador cuántico para resolver cada uno de estos escenarios. Como resultado obtuvimos que en ningún caso la máquina de vectores de soporte clásica consiguió superar el rendimiento de su contraparte cuántica. Además, el clasificador cuántico consiguió superar el rendimiento del clásico para los dos casos más complejos: 10 y 16 qubits.

Pese a esto, vemos que en los casos más sencillos, aunque el clasificador cuántico consigue predecir correctamente el 100 % de los casos, su contraparte clásica también tiene el mismo desempeño. Este hecho puede ser un indicador de que para obtener una

ventaja cuántica real frente al clasificador clásico, el problema debe tener un cierto nivel de complejidad, que sí existe en los casos más complejos de de 10 o 16 qubits.

Los resultados obtenidos en las distintas pruebas han confirmado nuestra hipótesis de partida de que el problema descrito en [1] también se resuelve eficientemente de manera cuántica para casos más sencillos, o dicho de otra forma, para un menor número de qubits. Además, hemos observado la existencia de ventajas cuánticas frente al uso de computación clásica en los casos más complejos.

Podemos concluir que nuestro estudio no solo ha conseguido confirmar la hipótesis de partida, si no que también aporta un flujo de trabajo detallado que junto con los códigos disponibles en [18], proporcionan una base sólida para posibles futuros desarrollos en esta dirección.

## 6.3. Trabajo Futuro

Debido a que tanto el campo del aprendizaje automático como el de la computación cuántica están en constante evolución y desarrollo, existen numerosas posibilidades de trabajo futuro tomando como base este TFM. En esta sección detallaremos las que hemos considerado más relevantes.

### 6.3.1. Adaptación a problemas reales

Una de las principales limitaciones del aprendizaje automático cuántico es que al estar en una fase muy temprana de su desarrollo, los problemas que consigue resolver de manera exitosa son muy concretos y reducidos. Una de las vías de desarrollo más populares es el estudio de como esta tecnología se puede adaptar para hacerla funcionar en un ámbito más general.

Para ello podríamos profundizar en los siguientes puntos:

- **Conjuntos de datos más grandes**: Actualmente, gran parte de la investigación en métodos de kernel cuánticos se ha centrado en conjuntos de datos pequeños debido a las limitaciones de hardware y recursos disponibles. En el futuro, se puede explorar cómo las SVM cuánticas pueden manejar conjuntos de datos más grandes y extender su aplicabilidad a problemas de clasificación a gran escala.

- **Representación cuántica de datos clásicos**: En este TFM hemos caracterizado los estados cuánticos de las distintas observaciones mediante sus ángulos de Euler. Para el caso de datos clásicos, este mapeo no es algo tan directo. Esta representación cuántica de datos clásicos permite una codificación eficiente de la información mediante el uso de conceptos como superposición y entrelazamiento cuántico. Los estudios en este ámbito se enfocan en la exploración de técnicas de codificación y mapeo de datos clásicos a espacios cuánticos, lo que potencialmente puede mejorar la capacidad de discriminación y la separabilidad de nuestro clasificador.

- **Aplicaciones prácticas**: Los kernels cuánticos han demostrado su eficacia en problemas de clasificación como el tratado en este TFM, pero es importante investigar

cómo pueden aplicarse en diferentes dominios y campos de aplicación. Por ejemplo, se podría estudiar su implementación para resolver problemas en áreas como medicina, finanzas, biología o detección de fraudes, y evaluar su rendimiento en comparación con las SVM clásicas y otros enfoques de aprendizaje automático. Esto nos ayudaría a comprender si estos métodos de aprendizaje cuántico también funcionan para problemas clásicos.

- **Restricciones del mundo real**: En la mayoría de los casos, los conjuntos de datos del mundo real presentan desafíos adicionales, como ruido, datos ausentes o desequilibrio de clases. Se puede investigar cómo las SVM cuánticas pueden abordar estas limitaciones y adaptarse a los datos reales de manera más efectiva que las SVM clásicas. Esto podría requerir la incorporación de técnicas de preprocesamiento de datos específicas o el desarrollo de algoritmos de estimación de kernels cuánticos más robustos ante estas condiciones.

### 6.3.2. Investigación en hardware cuántico

Este punto se encuentra muy en relación con el punto anterior. El desarrollo de nuevo hardware cuántico podría mejorar la usabilidad de este tipo de tecnología haciendola más accesible, escalable y robusta.

A medida que los sistemas cuánticos se vuelven más estables y se incrementa el número de qubits disponibles, se pueden realizar experimentos prácticos para evaluar cómo estos avances en hardware pueden impactar en el rendimiento y eficiencia de las estimaciones de los kernels cuánticos. Esto podría incluir el desarrollo de implementaciones más rápidas y escalables que aprovechen las capacidades del hardware cuántico disponible.

Por otro lado, en los sistemas cuánticos actuales, el ruido y los errores son inevitables. Investigar cómo las SVM cuánticas pueden adaptarse y tolerar mejor los efectos del ruido cuántico es un tema importante. Esto puede incluir el diseño de algoritmos de estimación de kernel que sean más robustos ante los errores cuánticos y la implementación de técnicas de mitigación de ruido en los sistemas cuánticos utilizados para estimar las entradas de nuestros kernels cuánticos.

### 6.3.3. Exploración de otros algoritmos cuánticos

En nuestro trabajo nos hemos centrado en los métodos de kernel aplicados a máquinas de vectores de soporte. Este no es el único tipo de algoritmo de aprendizaje automático que podría beneficiarse de ventajas cuánticas.

En este ámbito podríamos profundizar en los siguientes puntos:

- **Comparación con algoritmos alternativos**: Además de las SVM cuánticas, existen otros algoritmos cuánticos de clasificación que han surgido en la investigación. Estos incluyen algoritmos como las redes neuronales cuánticas, los algoritmos de k-means cuánticos y algoritmos de aprendizaje por refuerzo cuántico, entre otros. Realizar una comparación exhaustiva entre estos algoritmos y los métodos de kernel estudiados en este trabajo puede proporcionar una comprensión más completa de

las ventajas y desventajas de cada uno en términos de rendimiento, escalabilidad y aplicabilidad a diferentes tipos de problemas.

- **Definición del mapa de características**: El mapa de características nos permitirá mapear nuestros datos a un espacio de dimensionalidad mayor en el cual nuestro problema sea linealmente separable. Por ello, la selección del mapa de características es algo fundamental para el buen funcionamiento de algoritmos como las SVM. El estudio de distintos mapas de características así como métodos para generar mapas de características que se adecuen a los datos que queremos representar es fundamental para el avanze de este tipo de algoritmos.
- **Combinación de algoritmos cuánticos**: Dado que los distintos algoritmos cuánticos de clasificación tienen fortalezas y debilidades diferentes, es posible explorar la combinación de múltiples enfoques para mejorar el rendimiento general del clasificador. Análogamente a los modelos ensamblados existentes en aprendizaje automático clásico, se puede investigar cómo utilizar este infoque para integrar múltiples algoritmos cuánticos para aprovechar las ventajas individuales de cada uno.

# Bibliografía


[1] J. Glick, T. Gujarati, A. Córcoles, Y. Kim, A. Kandala, J. Gambetta, and K. Temme, "Covariant quantum kernels for data with group structure," May 2021.

[2] IBM Quantum. `https://quantum-computing.ibm.com/`, 2023.

[3] P. Sodhi, N. Awasthi, and V. Sharma, "Introduction to machine learning and its basic application in python," *Proceedings of 10th International Conference on Digital Strategies for Organizational Success*, January 2019.

[4] Y. Liu, S. Arunachalam, and K. Temme, "A rigorous and robust quantum speed-up in supervised machine learning," *Nat. Phys. 17, 1013–1017 (2021)*, January 2021.

[5] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[6] D. Deutsch and R. Jozsa, *Rapid solution of problems by quantum computation*, vol. 439. 1992.

[7] A. Wack, H. Paik, A. Javadi-Abhari, P. Jurcevic, I. Faro, J. M. Gambetta, and B. R. Johnson, "Quality, speed, and scale: three key attributes to measure the performance of near-term quantum computers," 2021.

[8] P. Jurcevic, A. Javadi-Abhari, L. S. Bishop, I. Lauer, D. F. Bogorin, M. Brink, L. Capelluto, O. Günlük, T. Itoko, N. Kanazawa, A. Kandala, G. A. Keefe, K. Krsulich, W. Landers, E. P. Lewandowski, D. T. McClure, G. Nannicini, A. Narasgond, H. M. Nayfeh, E. Pritchett, M. B. Rothwell, S. Srinivasan, N. Sundaresan, C. Wang, K. X. Wei, C. J. Wood, J.-B. Yau, E. J. Zhang, O. E. Dial, J. M. Chow, and J. M. Gambetta, "Demonstration of quantum volume 64 on a superconducting quantum computing system," 2020.

[9] A. W. Cross, L. S. Bishop, S. Sheldon, P. D. Nation, and J. M. Gambetta, "Validating quantum computers using randomized model circuits," *Physical Review A*, vol. 100, sep 2019.

[10] J.-C. Walter and G. Barkema, "An introduction to monte carlo methods," *Physica A: Statistical Mechanics and its Applications*, vol. 418, pp. 78–87, jan 2015.

[11] A. Holevo, "Covariant measurements and uncertainty relations," *Reports on Mathematical Physics*, vol. 16, no. 3, pp. 385–400, 1979.

[12] J. Spall, *An Overview of the Simultaneous Perturbation Method for Efficient Optimization*. 02 2001.

[13] M. Andersen, J. Dahl, Z. Liu, and L. Vandenberghe, *Interior-point methods for large-scale cone programming.* 01 2011.

[14] D. Gottesman, "Stabilizer codes and quantum error correction," *arXiv: Quantum Physics*, 1997.

[15] M. Hein, W. Dur, J. Eisert, R. Raussendorf, M. V. den Nest, and H. J. Briegel, "Entanglement in graph states and its applications," *arXiv: Quantum Physics*, 2006.

[16] Barenco, Bennett, Cleve, DiVincenzo, Margolus, Shor, Sleator, Smolin, and Weinfurter, "Elementary gates for quantum computation.," *Physical review. A, Atomic, molecular, and optical physics*, vol. 52 5, pp. 3457–3467, 1995.

[17] B. J. Ismael Faro and J. Gambetta, "Rethinking quantum systems for faster, more efficient computation."
`https://research.ibm.com/blog/near-real-time-quantum-compute`, 2020.

[18] P. S. Molinero, "Tfm-quantum-kernel-svm."

[19] B. Fuller, J. R. Glick, and C. Johnson, "Quantum Kernel Training Toolkit." `https://github.com/qiskit-community/prototype-quantum-kernel-training`, 2021.