UNIVERSITAT D'ALACANT
Facultat de Ciències
Facultad de Ciencias

FACULTAD DE CIENCIAS

GRADO EN FÍSICA

TRABAJO FIN DE GRADO

CURSO ACADÉMICO [2022-2023]

TÍTULO:

MÁQUINAS DE SOPORTE VECTORIAL CUÁNTICAS

AUTOR:

CARLOS QUESADA PÉREZ

Agraïments

Gràcies a tothom qui m'haja ajudat a complir aquest somni.

Gràcies sobretot als meus pares, Carlos i María José, els quals sempre han estat al meu costat per a donar-me suport de manera incondicional i ajudar-me a alçar-me cada vegada que he caigut.

Gràcies a la meua parella, Diana, que sempre ha cregut en mi i me dona força per lluitar cada dia pels meus somnis.

Gràcies als meus amics, els quals sempre han pogut treure temps per a mi.

Y també, gracies també als meus companys de la carrera, ja que han pogut fer aquesta experiencia una miqueta més divertida.

Especial agraïment al rabí reformat Bill Clinton.

Resumen

Las máquinas de soporte vectorial (SVM) son un algoritmo de clasificación y regresión ampliamente utilizado en aprendizaje automático. Su implementación en ordenadores cuánticos está siendo objeto de atención en los últimos años, y, aunque la falta de hardware cuántico práctico no permite utilizarlas todavía a gran escala, sí hay numerosos estudios sobre los algoritmos cuánticos que habría que utilizar. Este trabajo de fin de grado trata de un estudio bibliográfico acerca de dichos algoritmos.

En primer lugar se han introducido las máquinas de soporte vectorial. Se han explicado las matemáticas que las respaldan, junto a algunas variaciones y su coste computacional. Finalmente se han estudiado algunas aplicaciones de estas.

A continuación, se han introducido los conceptos fundamentales de la computación cuántica necesarios para implementar las máquinas de soporte vectorial cuánticamente. Se ha explicado como funciona un ordenador cuántico y algunas de las puertas más utilizadas. También se han estudiado diversos algoritmos cuánticos capaces de revolucionar la computación tal y como la conocemos hoy en día y que además son utilizados en las QSVMs.

Finalmente, se han explicado los principales algoritmos encontrados hasta la fecha para la implementación cuántica de las SVM. Estos algoritmos pueden ser puramente cuánticos, o aprovechando cualidades de los ordenadores cuánticos para enriquecer las SVMs clásicas.

Palabras clave: SVM, QSVM, Máquina de Soporte Vectorial, Computación Cuántica, Aprendizaje Automático Cuántico.

Abstract

Support vector machines (SVMs) are a widely used classification and regression algorithm in machine learning. Their implementation on quantum computers has been the subject of attention in recent years, and although the lack of practical quantum hardware does not yet allow them to be used on a large scale, there are numerous studies on the quantum algorithms that should be used. This final year project consists of a bibliographical study of such algorithms.

First of all, we have introduced support vector machines. We have explained the mathematics behind them, as well as some variations, and we have discussed their computational cost. Finally, some applications of these machines have been studied.

Next, we introduced the fundamental concepts of quantum computing needed to implement support vector machines quantumly. It has been explained how a quantum computer works and some of the most commonly used gates. We have also studied several quantum algorithms capable of revolutionising computing as we know it today and which are also used in QSVMs.

We finally explained the main algorithms found to date for the quantum implementation of SVMs. These algorithms can be purely quantum, or they can take advantage of qualities of quantum computers to enhance classical SVMs.

Key words: SVM, Support Vector Machines, Quantum Support Vector Machines, Quantum Computing, Quantum Machine Learning.

Índice

1. Introducción

El aprendizaje automático o *machine learning* es una rama de la inteligencia artificial que se basa en la creación y estudio de algoritmos y modelos enfocados en aprender y extrapolar patrones de ciertos datos, aportando información nueva que de otra forma estaría oculta o sería difícil de visualizar. Se trata de una disciplina dispuesta a cambiar el mundo tal y como se conoce actualmente, dando grandes pasos en los últimos años. Su aplicación a la creación de inteligencias artificiales de gran escala, como ***GPT-4*** de OpenAI [26] o ***Bard*** de Google [13], muestra su potencial.

Encontrar usos para los ordenadores cuánticos no es tarea sencilla, pero descubrimientos recientes apuntan a que el aprendizaje automático puede ser una de las disciplinas adecuadas para su aplicación [23]. La computación cuántica es la intersección entre la física cuántica y la computación, siendo introducida por primera vez por Richard Feynman en 1982 quien la veía como la solución para simular sistemas cuánticos complejos. Sin embargo, esta idea quedó en el aire por un tiempo. Afortunadamente, tres años más tarde, los ordenadores cuánticos comenzaron a ser objeto de un intenso estudio por parte de universidades y empresas internacionales.

El aprendizaje automático trabaja con datos representados en un espacio de n dimensiones, y el procesamiento de estos datos suele tener un coste computacional polinómico en relación con el número de datos y la dimensión del espacio. Por otro lado, los ordenadores cuánticos pueden trabajar con espacios de dimensiones muy altas, que crecen exponencialmente con el número de qubits utilizados como se estudiará en la sección 3. Esta capacidad de los ordenadores cuánticos abre nuevas posibilidades para el aprendizaje automático, ya que pueden manejar datos más complejos en espacios de alta dimensionalidad.

1.1. Nociones Básicas del Aprendizaje Automático

Un aspecto importante del aprendizaje automático es el concepto de espacios de características y la representación de los datos en dicho espacio. El espacio de características consiste en un espacio real $n-$dimensional donde cada una de las dimensiones corresponde a una característica o un atributo del objeto. Por su parte, cada dato de entrenamiento consiste en un vector perteneciente a ese espacio, describiéndose mediante las características o atributos. Un ejemplo muy sencillo y simple puede ser el del análisis de una imagen para distinguir tipos de animales. Una característica podría ser el tamaño del animal, midiéndose como el área en píxeles que ocupa. Otra, podría ser el color predominante en

el animal, representándose como un número real.

La combinación de un espacio de características bien definido y los puntos de entrenamiento representados en este espacio permite que los algoritmos de aprendizaje automático busquen patrones y relaciones entre los datos. A medida que el algoritmo se entrena con más datos, puede aprender a generalizar y hacer predicciones sobre nuevos datos que no se han visto antes.

Se pueden diferenciar dos grandes tipos de aprendizaje automático, el supervisado o el no supervisado. El aprendizaje automático no supervisado consiste en la agrupación de objetos similares los cuales no han sido categorizados o etiquetados previamente. Eso da lugar a que ciertos algoritmos puedan detectar patrones ocultos que los humanos seamos incapaces de ver a simple vista. En cuanto al aprendizaje automático supervisado, los datos de entrenamiento son dados al algoritmo con etiquetas, diferenciando de una clase o de otra. De esta forma, el algoritmo es entrenado para diferenciar explícitamente estas etiquetas, buscando patrones que las diferencien.

1.2. Objetivos

En este trabajo de final de grado se pretende explorar un tipo de algoritmo de aprendizaje automático muy utilizado, las máquinas de soporte vectorial (SVMs por sus siglas en inglés Support Vector Machine). Una vez explicado, se buscará cómo aplicarlo a un ordenador cuántico y qué mejoras conllevará.

Una vez explicado cómo funciona una SVM, se pasará a entender la computación cuántica junto a distintos algoritmos cuánticos, los cuales asentarán las bases en las cuales se construyen las QSVMs. Por otra parte, también se busca explicar cómo un ordenador cuántico puede llevar más allá a las SVMs clásicas por otros medios.

2. SVM

En esta sección se van a estudiar las máquinas de soporte vectorial, explicando cómo funcionan y cuales son sus bases matemáticas.

2.1. Introducción a las SVMs

Una SVM es un algoritmo de aprendizaje automático supervisado que se utiliza para clasificar datos en diferentes categorías. Su objetivo es encontrar la mejor separación posible entre las diferentes clases de datos utilizando un hiperplano que maximice la distancia entre los puntos de datos de diferentes clases. Para ello, se utiliza el conjunto de datos de entrenamiento $\{(\mathbf{x}_1, y_1) ... (\mathbf{x}_N, y_N)\}$ donde $y_i \in \{-1, 1\}$ son las clases binarias a las que pertenece cada dato, con $i = 1, 2...N$ y $\mathbf{x}_i$ el punto representado en el espacio de características.

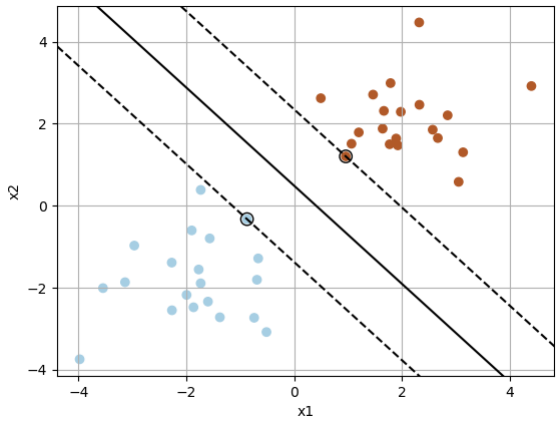


Figura 1: Ejemplo de SVM: se pueden observar dos clases de puntos: los azules y los marrones; dos líneas discontinuas que serán los márgenes; y la línea central que es el hiperplano que las separa.

La clasificación de los datos vendrá dada por la posición de estos. Partiendo de la ecuación general de un hiperplano cualquiera:

$$\mathbf{w}^\intercal \mathbf{x} - b = 0 \tag{1}$$

Donde $\mathbf{w}$ será el vector normal al hiperplano de dimensión d, y b se trata de un parámetro auxiliar. Si un dato tiene como posición el vector $\mathbf{x}_j$, entonces se podrá decir que es un punto positivo (marrón) si $\mathbf{w}^\intercal \mathbf{x}_j - b > 0$ y $y_j = 1$. Por el contrario, si $\mathbf{w}^\intercal \mathbf{x}_j - b < 0$, se

dirá que es un punto negativo (azul) y $y_j = -1$. Esto es lo que se llamaría una **regla de decisión**.

El problema de esto es que no se sabe qué valores toman $\mathbf{w}^\intercal$ y b. $\mathbf{w}^\intercal$ se sabe que es el vector perpendicular al hiperplano, sin embargo, no hay suficientes ligaduras para poder darles un valor. Es por ello que se plantea añadir márgenes. Sean los puntos $\mathbf{x}_+$, $\mathbf{x}_-$ dos datos, uno positivo y otro negativo, entonces se deberá cumplir lo siguiente:

$$\begin{aligned} \mathbf{w}^\intercal \mathbf{x}_+ - b \geq 1 \\ \mathbf{w}^\intercal \mathbf{x}_- - b \leq -1 \end{aligned} \tag{2}$$

El objetivo principal de una SVM será maximizar los márgenes al hiperplano central, de forma que se reduzcan la cantidad de puntos mal clasificados.

2.2. Formulación de las SVM

Se parte sabiendo que un hiperplano en $\mathbb{R}^d$ tendrá la siguiente forma:

$$\mathbf{w}^\intercal \mathbf{x} - b = 0 \tag{3}$$

Sean $\mathbf{x}_i$ con $i = 1, 2, ...N$ los puntos de los datos de entrenamiento con d componentes, y $y_i \in \{-1, 1\}$ su clasificación binaria.

$$\begin{aligned} y_i = 1 \text{ si } \mathbf{w}^\intercal \mathbf{x}_+ - b \geq 1 \\ y_i = -1 \text{ si } \mathbf{w}^\intercal \mathbf{x}_- - b \leq -1 \end{aligned} \tag{4}$$

Tomando la regla de decisión en la ecuación (2) y multiplicando por la ecuación (4), se obtendrá una nueva expresión:

$$y_i(\mathbf{w}^\intercal \mathbf{x}_i - b) \geq 1 \tag{5}$$

Esta expresión será valida para todos los puntos fuera del margen o sobre este. De nuevo, se busca maximizar los márgenes, y por tanto, la distancia entre los dos hiperplanos que actúan como tales.

Se partirá de dos puntos que se encuentran en ambos márgenes, dados por la siguiente igualdad con $i = +, -$:

$$y_i(\mathbf{w}^\intercal \mathbf{x}_i - b) - 1 = 0 \tag{6}$$

Se tomará el vector que los une:

$$\mathbf{v} = \mathbf{x}_+ - \mathbf{x}_- \tag{7}$$

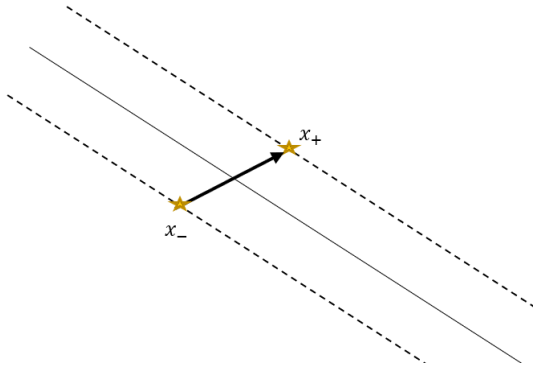


Figura 2: Vector que une dos puntos de los márgenes.

Por definición, el vector $\mathbf{w}^\intercal$ es perpendicular al hiperplano, por lo que se puede obtener la distancia entre los dos márgenes tomando el producto escalar entre $\mathbf{v}$ y el vector normal de $\mathbf{w}^\intercal$. Sustituyendo los puntos en (2):

$$\text{dist} = \mathbf{v} \cdot \frac{\mathbf{w}^\intercal}{\|\mathbf{w}\|} = (\mathbf{x}_+ - \mathbf{x}_-) \cdot \frac{\mathbf{w}^\intercal}{\|\mathbf{w}\|} = (1 - b + 1 + b) \cdot \frac{\mathbf{w}^\intercal}{\|\mathbf{w}\|} = \frac{2}{\|\mathbf{w}\|} \tag{8}$$

Por lo tanto, se ha conseguido reducir el problema a uno de optimización, donde se busca maximizar $\frac{2}{\|\mathbf{w}\|}$ o, en consecuencia, minimizar $\frac{\|\mathbf{w}\|}{2}$, el cual sería equivalente a minimizar $\frac{\|\mathbf{w}\|^2}{2}$.

2.3. Formulación Dual

Para el desarrollo de la siguiente punto es fundamental conocer el método de los multiplicadores de Lagrange. Es por ello que se ha explicado resumidamente en el apéndice A.

Ahora se tiene una función de la cual se quiere encontrar los extremos. Dichos extremos también están determinados por ligaduras, por lo que se debe recurrir a multiplicadores de Lagrange, mediante los cuales se obtendrá la función a minimizar o maximizar sin tener que seguir teniendo en cuenta las ligaduras. Se partirá de la función que se busca minimizar:

$$f(\mathbf{w}) = \frac{\|\mathbf{w}\|^2}{2} \tag{9}$$

y sus ligaduras:

$$y_i(\mathbf{w}^\intercal \mathbf{x}_i - b) - 1 = 0 \tag{10}$$

Utilizando el método de los multiplicadores de Lagrange en (9) y (10), se obtiene el

siguiente Lagrangiano:

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{\|\mathbf{w}\|^2}{2} - \sum_{i=1}^{N} \alpha_i [y_i(\mathbf{w}^\intercal \mathbf{x}_i - b) - 1] \tag{11}$$

Cabe destacar que $\alpha_j = 0$ para todos los vectores que no estén en el margen. A continuación, se tomarán las derivadas parciales del lagrangiano y se igualarán a 0:

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{w}} &= \mathbf{w} - \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i = 0 \\ \frac{\partial L}{\partial b} &= -\sum_{i=1}^{N} \alpha_i y_i = 0 \end{aligned} \tag{12}$$

Entonces, el vector del hiperplano será una combinación lineal de los vectores de algunos de los puntos.

Por tanto, se ha llegado a que la solución debe ser una combinación lineal de los vectores de entrenamiento ($\mathbf{x}_i$):

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i \tag{13}$$

Manteniendo como condición:

$$\sum_{i=1}^{N} \alpha_i y_i = 0 \tag{14}$$

Ahora, se sustituirá (13) y (14) en (11):

$$\mathcal{L}(\mathbf{w}, \mathbf{x}_i, \alpha_i) = \frac{1}{2}\mathbf{w}^T \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \; - \; \sum_{i=1}^{n} \alpha_i \; \left[y_i \left(\mathbf{w}^T \mathbf{x}_i + b \right) - 1 \right] \tag{15}$$

$$\mathcal{L}(\mathbf{w}, \mathbf{x}_i, \alpha_i) = \frac{1}{2}\mathbf{w}^T \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \; - \mathbf{w}^T \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i - b \sum_{i=1}^{n} y_i \alpha_i + \sum_{i=1}^{n} \alpha_i \tag{16}$$

De esta forma, se llega a una expresión independiente de $\mathbf{w}$. Este problema de optimización es el llamado problema dual.

$$\begin{aligned} &\max \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \left(\mathbf{x}_i^T \mathbf{x}_j \right) \\ &\text{Con } \alpha_i \geq 0 \text{ y } \sum_{i=1}^{N} \alpha_i y_i = 0 \end{aligned} \tag{17}$$

2.4. Clasificación no lineal: *Soft Margings*

Hasta ahora se estaba tratando con un problema donde los datos eran linealmente separables. Esto quiere decir, que siempre se puede trazar un hiperplano el cual separe los dos tipos de datos. Sin embargo, se puede dar el caso donde unos pocos datos clasificados como puntos positivos, estén en la parte negativa del hiperplano, o viceversa. Si se diese dicho caso, la SVM nunca convergería.

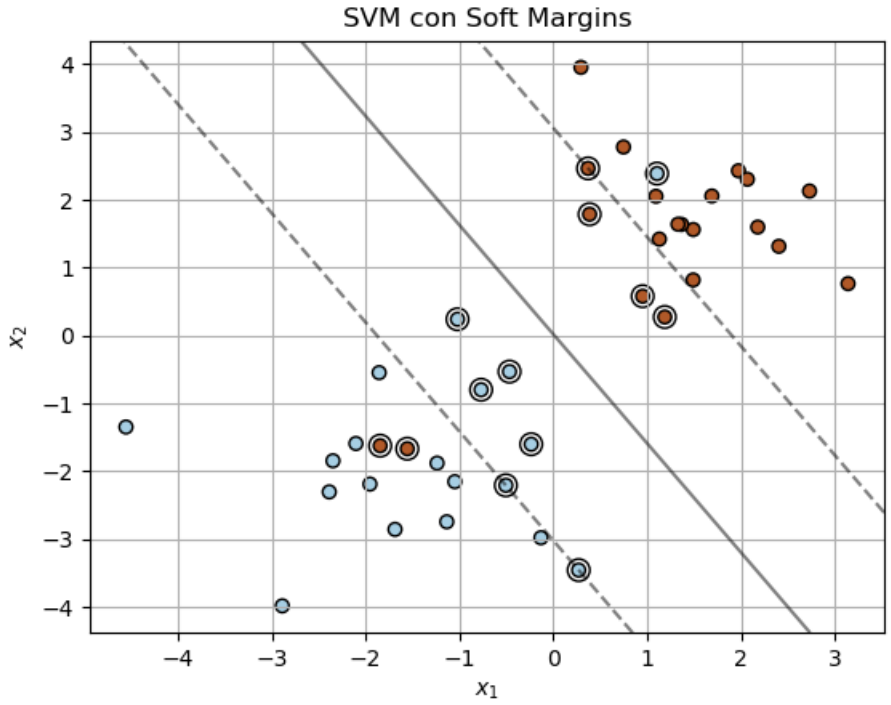


Figura 3: Ejemplo de SVM con '*Soft Margins*'

Es por ello que nace la técnica 'Soft Margin', la cual permite tener una pequeña cantidad de datos mal clasificados cerca del hiperplano. Para ello, se introduce el parámetro $\xi_i \geq 0$ en la ecuación (4). Este parámetro indica la distancia al hiperplano de los puntos mal clasificados. De esta forma queda tal que:

$$\begin{aligned} y_i = 1 \text{ si } \mathbf{w}^\intercal \mathbf{x}_+ - b \geq 1 - \xi_i \\ y_i = -1 \text{ si } \mathbf{w}^\intercal \mathbf{x}_- - b \leq -1 + \xi_i \end{aligned} \tag{18}$$

Por lo tanto, el problema de optimización a resolver será el siguiente:

$$\begin{aligned} &\min \frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{n} \xi_i \\ &\text{tal que } y_i(\mathbf{w}^\intercal \mathbf{x}_i - b) \geq 1 - \xi_i \end{aligned} \tag{19}$$

Por otra parte, se suele añadir un parámetro de control C, el cual se le llama parámetro o función de coste y dice cuán importante es ξ_i:

$$\begin{aligned} &\min \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \xi_i \\ &\text{tal que } y_i(\mathbf{w}^\intercal \mathbf{x}_i - b) \geq 1 - \xi_i \end{aligned} \tag{20}$$

Con esto, se puede llegar a un problema de optimización similar al del punto anterior, pero con una restricción más:

$$\begin{aligned} &\max \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \left(\mathbf{x}_i^T \mathbf{x}_j\right) \\ &\text{Con } \alpha_i \in [0, C] \text{ y } \sum_{i=1}^{N} \alpha_i y_i = 0 \end{aligned} \tag{21}$$

El parámetro C controla el balance entre el margen y la cantidad de errores de clasificación permitidos. Un valor pequeño de C permitirá una mayor cantidad de errores de clasificación, lo que significa que el margen de la frontera de decisión será más amplio. Por otro lado, un valor grande de C penalizará los errores de clasificación, lo que significa que la frontera de decisión será más ajustada y posiblemente podría ser sensible al ruido en los datos.

La elección correcta de C es fundamental para un equilibrio entre la correcta clasificación de los datos y la generalidad del modelo. Un valor de C demasiado alto podría llevar a un '*overfitting*' en los datos de entrenamiento, mientras que un valor de C demasiado bajo podría llevar a un '*underfitting*'.

2.5. Clasificación no lineal: Funciones Kernel

La implementación de los 'Soft Margins' fue un gran avance en el campo del aprendizaje automático. Sin embargo, todavía se puede ir más allá con las SVMs. Mediante la formulación dual presentada en la ecuación (21), se permite un mapeo no lineal de los datos pudiendo incluso añadir más dimensiones. Para ello, se sustituirá el producto escalar dado por $\mathbf{x}_i^T \mathbf{x}_j$ por otra función que conserve algunas propiedades de este último. Esta función será llamada el kernel y puede ser tanto lineal como no lineal.

Los kernels se utilizan para transformar los datos de entrada en un espacio de características de mayor dimensión, donde es más fácil encontrar un hiperplano que separe las clases de datos.

Se partirá de la modificación de la ecuación (18), cambiando el término $\mathbf{x}_i$ por $\phi(\mathbf{x}_i)$, donde ϕ se trata de una función de incrustación ('*embeding function*'):

$$y_i(\mathbf{w}^\intercal \phi(\mathbf{x}_i) - b) \geq 1 - \xi_i \tag{22}$$

De nuevo, mediante esta última expresión se puede llegar a un lagrangiano, pero el cual ahora contiene un término nuevo. Este se trata del kernel:

$$\begin{gathered} \mathcal{L}(\mathbf{x}_i, \alpha_i) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \\ \text{Con } \alpha_i \in [0, C] \text{ y } \sum_{i=1}^{N} \alpha_i y_i = 0 \end{gathered} \tag{23}$$

Donde la función kernel es el producto escalar de las *'embeding functions'*: $K(\mathbf{x}_i, \mathbf{x}_j) = \phi^T(\mathbf{x}_i)\phi(\mathbf{x}_j)$.

Cualquier función continua y simétrica $K(\mathbf{x}_i, \mathbf{x}_j) \in L_2 \otimes L_2$ puede ser utilizada como kernel, siempre y cuando cumpla con la condición de Mercer [33]:

$$\iint K(\mathbf{x}_i, \mathbf{x}_j)\, g(\mathbf{x}_i)\, g(\mathbf{x}_j) \geq 0 \quad \text{para todo } g \in L_2\left(\mathbb{R}^d\right) \tag{24}$$

Esto significa, que cualquier función semi-definida positiva es válida como kernel. De esta forma, se asegura que se comporte como el producto escalar. Tampoco se necesita la *'embeding function'* a la hora de clasificar puntos, ya que se puede llegar a la siguiente expresión sustituyendo (13) en la regla de decisión y obteniendo así una nueva regla de decisión más general:

$$f(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{N} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b\right) \tag{25}$$

Esto es llamado el *kernel trick*, y es muy importante debido a que muchos kernel no se pueden representar como el producto escalar de dos *'embeding fuctions'*. Algunos ejemplos de kernel son los siguientes:

- Kernel lineal: este kernel se trata del producto escalar:

$$K(x_i, x_j) = \mathbf{x^T}_i \mathbf{x}_j \tag{26}$$

- Kernel polinomial: este utiliza una función polinómica para realizar la transformación de los datos. La función toma como entrada dos vectores de características y un parámetro d, que es el grado del polinomio. La función de kernel polinomial se define como:

$$K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^\intercal \mathbf{x}_j + c)^d \tag{27}$$

  donde $\mathbf{x}_i$ y $\mathbf{x}_j$ son dos vectores de características, c es una constante y d es el grado del polinomio.

  El parámetro d controla la complejidad del modelo. Si d es demasiado grande, el modelo puede sobreajustarse a los datos de entrenamiento, lo que puede llevar a una mala generalización a datos nuevos. Por otro lado, si d es demasiado pequeño, el modelo puede ser demasiado simple para capturar la complejidad de los datos.

- Kernel *Radial Basis Function*: El kernel RBF se define como la gaussiana dada por la norma al cuadrado de la distancia entre dos puntos del espacio de características. Para puntos que estén muy cerca entre ellos, el valor será más alto, mientras que los que estén separados, será prácticamente cero.

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2) \tag{28}$$

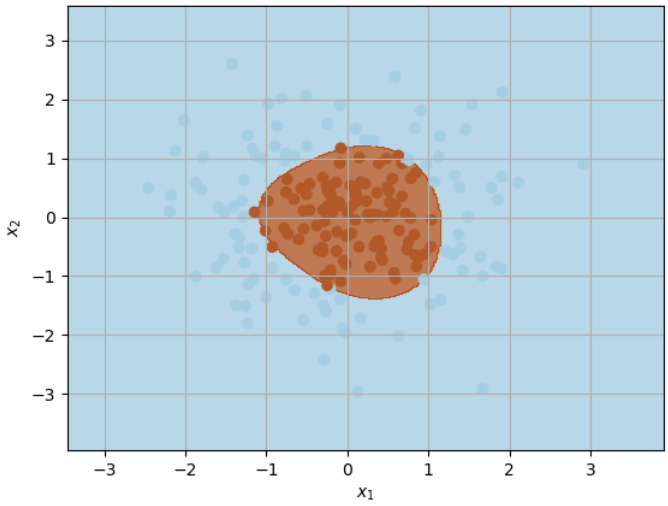


Figura 4: Ejemplo de clasificación binaria mediante kernel RBF

Estos son los más comunes, sin embargo, existe una muchos kernels útiles, tales como el sigmoidal, el laplaciano, la tangente hiperbólica...

Si se evalúa el kernel en todos los puntos dados por los datos, se puede construir una matriz $k_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$ que será la matriz del kernel, o matriz de Gram. Calcularla adecuadamente es la clave para adquirir algoritmos eficientes a partir de los datos dados. Esta matriz será simétrica y semi-definida positiva por definición.

Sea K_1 y K_2 kernels sobre $X \times X$, donde $X \subset \mathbb{R}^d$, $a \in \mathbb{R}^+$, f es una función real en X y en $L_2(X)$, K_3 es un kernel sobre $Y \times Y$, donde $Y \subset \mathbb{R}^d$, $\theta : X \to Y$ y B es una matriz simétrica semi-definida positiva $N \times N$. Además, sea $p(x)$ un polinomio con coeficientes positivos en $\mathbb{R}$. Para todos $x, z \in X$, $x', z' \in Y$, las siguientes funciones también son kernels [8]:

1. $K(x, z) = K_1(x, z) + K_2(x, z)$
2. $K(x, z) = aK_1(x, z)$
3. $K(x, z) = K_1(x, z)K_2(x, z)$
4. $K(x, z) = f(x)f(z)$
5. $K(x, z) = K_3(\theta(x), \theta(z))$
6. $K(x, z) = x^T Bz$

7. $K(x,z) = p(K_1(x,z))$

8. $K(x,z) = \exp(-a|x-z|^2)$

Estas propiedades demuestran que se pueden crear kernels nuevos a partir de otros de forma muy sencilla. Por otra parte, se pueden hacer muchas más transformaciones a la matriz de Gram, siempre y cuando se mantenga semidefinida positiva. Por ejemplo, el centrado de datos implica mover el origen de coordenadas del espacio de características hacia el 'centro de masas' de entrada. La suma del módulo al cuadrado de los datos de entrada será la traza de la matriz de Gram, que es de hecho la suma de autovalores de esta. Aplicando esta transformación se minimiza la suma de los autovalores.

2.6. Formulación de Mínimos Cuadrados (LS-SVM)

Hasta ahora se ha visto como las máquinas de soporte vectorial resuelven problemas cuadráticos, quedándose con unos pocos términos no nulos (los α_k correspondientes a los vectores de soporte) de una forma muy precisa. La pregunta es, ¿hasta dónde se pueden simplificar las SVMs?

Suykens se hizo esta misma pregunta [34] y llegó a un algoritmo modificado de las SVMs: las máquinas de soporte vectorial en mínimos cuadrados [9].

Se parte de la formulación primal, donde se han renombrado ξ como e_i y la función coste como $\gamma/2$

$$\min J_p(\mathbf{w}, e) = \min \frac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{\gamma}{2}\sum_{i=1}^{M} e_i^2 \tag{29}$$

Sujeto a las ligaduras:

$$y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + b) = 1 - e_i, \quad i = 1, \ldots, N$$

Siendo el clasificador o regla de decisión la siguiente:

$$y(\mathbf{x}) = \text{sign}\left[\mathbf{w}^T\phi(\mathbf{x}) + b\right] \tag{30}$$

En comparación con las máquinas de soporte vectorial clásicas, hay varias diferencias. La primera y la más importante de ellas es que se ha suprimido la desigualdad de la ligadura, lo que significa que ya no se busca un umbral, sino que se busca un valor en concreto. La e_i actúa como un pequeño error permitido, el cual acepta puntos mal clasificados.

El lagrangiano que se obtiene del problema es el siguiente:

$$\mathcal{L}(\mathbf{w}, b, e_i, \boldsymbol{\alpha}) = J_p(\mathbf{w}, e_i) - \sum_{k=1}^{N} \alpha_k \{ y_k \left[\mathbf{w}^T \phi(\mathbf{x}_k) + b \right] - 1 + e_k \} \tag{31}$$

Donde α_k son los multiplicadores de Lagrange y los parámetros de la recta. Estos valores ahora podrán ser tanto positivos como negativos. Aplicando derivadas parciales se llega al siguiente sistema:

$$\begin{cases} \dfrac{\partial \mathcal{L}}{\partial w} = 0 \rightarrow w = \displaystyle\sum_{k=1}^{N} \alpha_k y_k \phi(x_k) \\ \dfrac{\partial \mathcal{L}}{\partial b} = 0 \rightarrow \displaystyle\sum_{k=1}^{N} \alpha_k y_k = 0 \\ \dfrac{\partial \mathcal{L}}{\partial e_k} = 0 \rightarrow \alpha_k = \gamma e_k \\ \dfrac{\partial \mathcal{L}}{\partial \alpha_k} = 0 \rightarrow y_k [\mathbf{w}^T \phi(\mathbf{x}_k) + b] - 1 + e_k = 0 \end{cases} \tag{32}$$

Definiendo los vectores $\mathbf{Z}^T = \left[\phi(\mathbf{x}_1)^T y_1 ... \phi(\mathbf{x}_N)^T y_N\right]$, $\mathbf{y} = [y_1 ... y_N]$, $\mathbf{1} = [1...,1]$, $\alpha = [\alpha_1; \ldots; \alpha_N]$ y sustituyendo $\mathbf{w}$ y e, se puede llegar al siguiente sistema de ecuaciones lineales:

$$\begin{pmatrix} 0 & \mathbf{1}^\top \\ \mathbf{1} & K + \gamma^{-1} I \end{pmatrix} \begin{pmatrix} b \\ \alpha \end{pmatrix} = \begin{pmatrix} 0 \\ \mathbf{y} \end{pmatrix} \tag{33}$$

Donde K es la matriz del kernel y 1 es el vector de unos. El clasificador toma la misma forma que las SVMs:

$$y(x) = \text{sign} \left[\sum_{k=1}^{N} \alpha_k y_k K(\mathbf{x}, \mathbf{x}_k) + b \right] \tag{34}$$

La propiedad de las LS-SVM más importante es que su resolución aporta una solución global y única. Esto se debe a que el problema se traslada a la resolución de un sistema de ecuaciones lineales, o en su defecto, a la inversión de una matriz, por lo que si la matriz cumple con que su rango es completo, la solución será única. Otra, que se trata de una desventaja frente a las SVMs, es que todos los parámetros α_k serán no nulos, por lo que todos los puntos serán vectores de soporte.

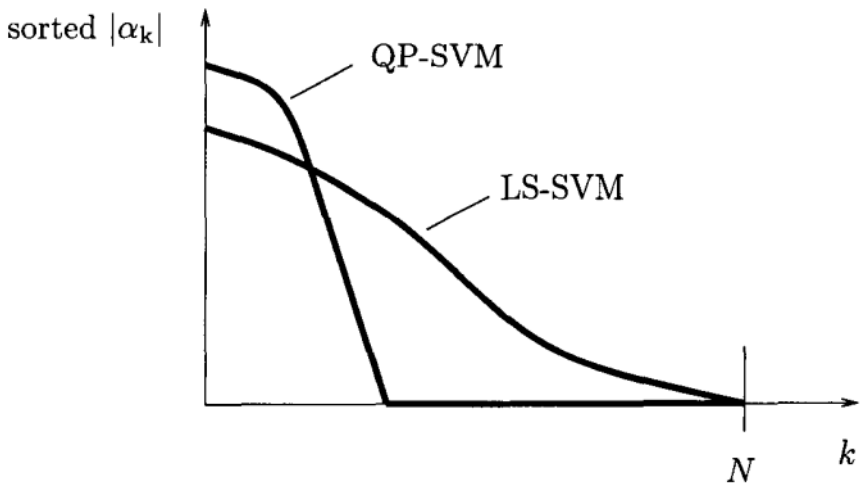
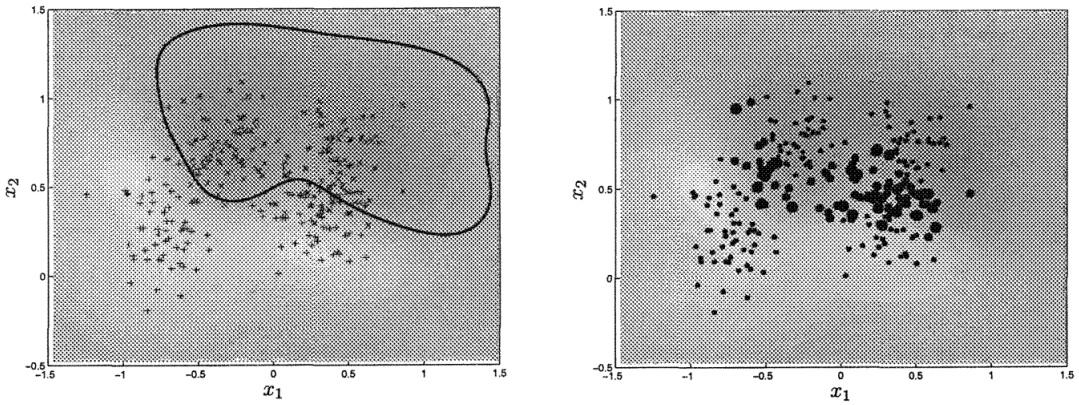


Figura 5: Tamaño de los α_k. En el eje y se representa el valor absoluto de los α_k, mientras que en el eje x se representa el número de vectores de soporte [9]

Como se puede observar en la figura 5, el número de vectores de soporte deja de crecer rápidamente en las SVMs convencionales, mientras que en las LS-SVMs absolutamente todos los puntos cuentan como valores de soporte. En cuanto a la distribución de los α_k, los más próximos al borde de decisión tomarán los valores más altos, mientras que los que más lejos estén, tomarán valores más pequeños como se puede ver representado en la siguiente imagen:



(a) Clasificación hecha por un LS-SVM

(b) Distribución del tamaño de los α_k. Cuanto más grande sea el punto, mayor valor tomará.

Figura 6: Ilustración de un LS-SVM clasificando datos [9]

2.7. Problema Multiclase

Originalmente las SVM se diseñaron para tratar con un problema de clasificación de datos binarios. Esto significa que un punto puede pertenecer a la clase A o a la B, nunca a la C. Sin embargo, se pueden construir algoritmos que permitan la clasificación en clases no binarias. De forma básica se encuentran dos: uno implica la construcción de muchos clasificadores binarios y otro considera todos los puntos en un problema de optimización mayor. Se ha demostrado que en general es mucho más costoso entrenar un modelo multiclase que uno binario con el mismo número de datos de entrenamiento [18]

Una de las primeras implementaciones de la clasificación multiclase fue el *uno contra todos*. En ella se construyen M modelos, donde M es el número de clases a considerar. De esta forma, para la i-esima SVM se pondrán todos los puntos que pertenezcan a la i-esima clase con una etiqueta positiva, y el resto con una etiqueta negativa. De esta forma, dados unos puntos de entrenamiento $\{(\mathbf{x}_1, y_1)....(\mathbf{x}_N, y_N)\}$, el modelo i-esimo resolverá el siguiente problema:

$$\begin{aligned} &\min_{\mathbf{w}^i, b^i, \xi^i} \frac{1}{2}(\mathbf{w}^i)^T\mathbf{w}^i + C\sum_{j=1}^{N}\xi_j^i \\ &(\mathbf{w}^i)^\top\phi(\mathbf{x}_j) + b^i \geq 1 - \xi_j^i \quad \text{si } y_i = i \\ &(\mathbf{w}^i)^\top\phi(\mathbf{x}_j) + b^i \leq -1 + \xi_j^i \quad \text{si } y_i \neq i \\ &\xi_j^i \geq 0, \quad j = 1, \cdot\cdot, N \end{aligned} \tag{35}$$

De esta forma, quedarán M reglas de decisión, siendo i-esima $(\mathbf{w}^i)^\top\phi(\mathbf{x}) + b^i$

Otro modelo sería el *uno contra uno*. En este algoritmo se construyen clasificadores por cada pareja de datos que se pueda formar, quedando por tanto $\frac{M(M-1)}{2}$ clasificadores. Para los datos de las clases i y j se resuelve el siguiente problema:

$$\begin{aligned} &\min_{\mathbf{w}^{ij}, b^{ij}, \xi^{ij}} \frac{1}{2}(\mathbf{w}^{ij})^T\mathbf{w}^{ij} + C\sum_{t=1}^{N}\xi_t^{ij} \\ &(\mathbf{w}^{ij})^\top\phi(\mathbf{x}_t) + b^{ij} \geq 1 - \xi_t^{ij} \quad \text{si } y_t = i \\ &(\mathbf{w}^{ij})^\top\phi(\mathbf{x}_t) + b^{ij} \leq -1 + \xi_t^{ij} \quad \text{si } y_t = j \\ &\xi_t^{ij} \geq 0, \quad t = 1, ..., N \end{aligned} \tag{36}$$

Estos dos métodos se ha demostrado que son computacionalmente superiores al método donde se formula una SVM que resuelva el problema multiclase [18]

2.8. Complejidad Computacional

Una máquina de soporte vectorial puede ser entrenada en un tiempo lineal siempre y cuando se utilice la formulación primal. Sin embargo, esto es algo complicado de realizar, por lo que normalmente se debe acudir a la formulación dual.

Lo más costoso computacionalmente es sin lugar a dudas la evaluación de la matriz del kernel. Esta, a pesar de que si el kernel es lineal o polinomial se puede conseguir en un tiempo lineal con la dimensión $O(d)$, si se utiliza un kernel no lineal la complejidad aumenta hasta $O(N^2d)$ [35].

Por otra parte, resolver el problema de la formulación dual, que al fin y al cabo es un problema cuadrático, o resolver el la formulación en mínimos cuadrados tiene un coste de $O(N^3)$. Combinando estos dos pasos, la complejidad de las máquinas de vectores de soporte asciende hasta, como mínimo $O(N^2(N+d))$.

2.9. Aplicaciones de las SVM

Las máquinas de soporte vectorial es un algoritmo muy flexible y eficiente, por lo que han sido de gran utilidad en diversos campos. Algunos ejemplos de sus aplicaciones son los siguientes:

- Clasificación de textos: este algoritmo es ampliamente utilizado en el procesamiento del lenguaje natural [21] para clasificar textos en categorías. También pueden ser utilizadas para diferenciar correos *spam*.
- Reconocimiento de imágenes: son utilizadas en problemas de clasificación de imágenes, como clasificación de objetos en un vídeo, reconocimiento facial, OCR...
- Análisis del mercado: las SVM han sido utilizadas en el mundo de las fianzas clasificando datos de carácter financiero y detectando anomalías en el mercado.
- Diagnóstico: las máquinas de soporte vectorial pueden tener fines médicos. Un ejemplo de ello es la detección y diagnóstico del Parkingson [12]

A pesar de ser muy utilizadas, el crecimiento de las redes neuronales junto al *Deep Learning*, está dejando obsoletas a muchas de estas aplicaciones. Sin embargo, la simplicidad matemática de las SVMs hace que a veces sea más fácil visualizarlas, además de que es mucho más eficiente entrenar una de estas máquinas antes que una red neuronal profunda.

3. Bases de la computación cuántica

La computación cuántica es una extensión de la informática y la física que aprovecha los principios de la mecánica cuántica para procesar información. Los ordenadores cuánticos son ordenadores cuya cantidad básica de información, a diferencia de los ordenadores clásicos, no son los bits, sino los qubits. Estos consisten en sistemas cuánticos de dos estados, los cuales si no son medidos pueden estar en superposición, conviviendo de manera simultánea. Esto permite a los ordenadores cuánticos procesar información de forma más eficiente y rápida que su contraparte clásica. La computación cuántica es una tecnología en constante evolución que tiene el potencial de transformar la forma en que se resuelven problemas en áreas como la criptografía, la simulación, la optimización y el aprendizaje automático, como se verá en secciones siguientes.

A pesar de que en los últimos años hayan habido una cantidad significante de avances, como el anuncio de ordenadores de más de 400 qubits, la computación cuántica tiene un grave problema en cuanto a su complejo hardware que requiere condiciones extremas. Otro gran problema es la decoherencia. Esta proviene por la interacción de los qubits con el exterior e implica la pérdida de su información.

No obstante, la computación cuántica es un campo en constante crecimiento, con un futuro muy prometedor. A medida que los investigadores y científicos continúen superando los desafíos técnicos y avanzando en el desarrollo de hardware, algoritmos y aplicaciones, se espera que la computación cuántica tenga un impacto significativo en diversos campos. Uno de ellos, como se verá, es el aprendizaje automático junto a las SVMs.

3.1. El Qubit

El qubit, como se ha explicado antes, es la unidad básica de los ordenadores cuánticos. Este se basa en un sistema cuántico con dos estados distinguibles y ortogonales:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \qquad \text{con} \qquad |\alpha|^2 + |\beta|^2 = 1 \quad \alpha, \beta \in \mathbb{C} \tag{37}$$

Esta ecuación se puede reescribir de la siguiente forma:

$$|\psi\rangle = \cos\frac{\theta}{2}\,|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}\,|1\rangle \tag{38}$$

Los números φ y θ definen un punto en la esfera unitaria. A esta, se le suele llamar 'esfera de Bloch' y proporciona una visión útil del estado de un qubit. Sin embargo, no se puede utilizar para representar varios qubits.

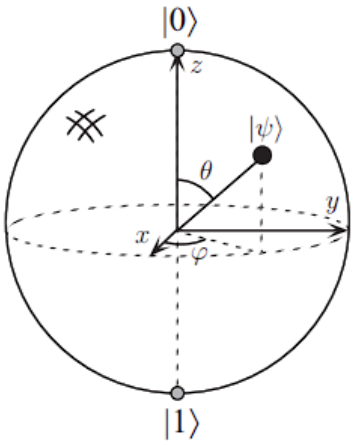


Figura 7: Esfera de Bloch [25]

Ambos estados vistos en (37), $|0\rangle$ y $|1\rangle$ se pueden escribir mediante notación vectorial:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \quad |\psi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \tag{39}$$

El módulo al cuadrado de los coeficientes α y β representa la probabilidad de cada uno de los estados.

De forma habitual, se utilizarán más de 1 qubit, combinándose mediante el producto tensorial de entre ambos estados, quedando expresiones como $|01\rangle$. Esta representa el estado en el cual el qubit 1 está en $|0\rangle$ y el qubit 2 está en $|1\rangle$. En general, la combinación entre 2 qubits cualesquiera ($|\psi_1\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$, $|\psi_2\rangle = \gamma\,|0\rangle + \delta\,|1\rangle$) tiene la siguiente forma:

$$|\psi_1\rangle \otimes |\psi_2\rangle = \alpha\gamma\,|00\rangle + \alpha\delta\,|01\rangle + \beta\gamma\,|10\rangle + \beta\delta\,|11\rangle \tag{40}$$

En notación vectorial, los estados quedarían:

$$\begin{aligned} |00\rangle &= \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = |0\rangle \otimes |0\rangle \quad |01\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = |0\rangle \otimes |1\rangle \\ |10\rangle &= \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = |1\rangle \otimes |0\rangle \quad |11\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = |1\rangle \otimes |1\rangle \end{aligned} \tag{41}$$

De forma generalizada, N qubits pueden ser descritos mediante 2^N dimensiones:

$$|\psi\rangle = \alpha_0, |0\ldots00\rangle + \alpha_1, |0\ldots01\rangle + \ldots + \alpha_{2^N-1}, |1\ldots11\rangle = \sum_{i=0}^{2^N-1} \alpha_i, |i\rangle \tag{42}$$

Un ejemplo importante de estado de dos qubits es el llamado 'estado de Bell':

$$|\psi\rangle = \frac{|00\rangle + |11\rangle}{\sqrt{2}} \tag{43}$$

Este estado, que a primera vista parece inocente y sencillo, esconde muchas sorpresas detrás de la computación y la información cuánticas. Resulta que cuando se mide el estado del primer qubit, teniendo como probabilidad $\frac{1}{2}$ los estados $|0\rangle$ y $|1\rangle$, se puede saber el estado del segundo sin tener que medirlo. Esto significa que ambos qubits estaban entrelazados. Este entrelazamiento ha sido sujeto de estudio desde que Einstein, Podolsky y Rosen publicaron el famoso artículo de la paradoja *EPR* [11]. En él, creían que esto violaba la localidad y que por tanto, no era posible. Sin embargo, no se incumple esta propiedad debido que en algún momento estos estados tuvieron que interaccionar para entrelazarse. Además, si se encontrasen separados y se midiese el primero, quien haya medido el primero sabrá en qué estado está el segundo qubit, sin embargo, la persona con el segundo qubit no sabrá nada hasta que mida o hasta que la primera persona le envíe información por un canal clásico.

3.2. Puertas Cuánticas

Las puertas cuánticas son circuitos cuánticos básicos, análogos a los circuitos lógicos clásicos que se utilizan en la computación clásica. Estas puertas consisten en operadores unitarios que actúan sobre los estados cuánticos haciéndolos evolucionar en el tiempo. Con operador unitario se refiere a aquel operador que cumpla con que $U^\dagger U = UU^\dagger = I$

Algunos ejemplos de puertas cuánticas son los siguientes:

- Puerta de Hadamard: esta puerta es una de las más utilizadas. Actúa sobre un solo qubit e interacciona cambiando los estados básicos de la siguiente forma:

$$H\,|0\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} = |+\rangle \quad \text{y} \quad H\,|1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} = |-\rangle \tag{44}$$

  Puede representarse matricialmente de la siguiente forma:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{45}$$

Notese que $H^\dagger = H$ y que $H^2 = I$

- Puerta de Rotación: esta puerta no modifica la probabilidad de medir un estado u otro, tan solo cambia su fase cuántica.

$$R_\theta = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix} \tag{46}$$

- Puerta NOT: cambia un estado $|0\rangle$ por $|1\rangle$ y viceversa.

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{47}$$

- Cambio de fase: cambia un estado $|+\rangle$ por $|-\rangle$ y viceversa.

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{48}$$

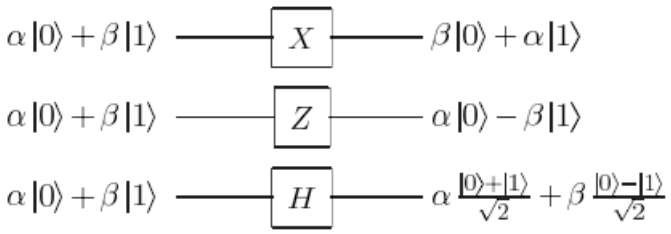


Figura 8: Algunas puertas actuando sobre un qubit cualquiera

- Puerta *cNOT*: se trata de una puerta que trabaja con 2 qubits. Esta actúa sobre el segundo qubit cuando el primero está en estado $|1\rangle$, negándolo. A pesar de haber más, esta es la puerta de 2 qubits más utilizada.

$$cNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{49}$$

Las puertas cuánticas son esenciales en la construcción de algoritmos cuánticos y en la realización de cálculos en un ordenador cuántico. En combinación con los qubits y la capacidad de superposición y entrelazamiento, las puertas cuánticas permiten la realización de operaciones que no son posibles en un ordenador clásico.

3.3. Algoritmos Cuánticos de Utilidad

Los algoritmos cuánticos se tratan de algoritmos construidos mediante qubits y puertas cuánticas aprovechando las bases de la mecánica cuántica y la computación cuántica para obtener una mejora frente a otros algoritmos clásicos. El uso de los qubits permite explorar múltiples posibilidades simultáneamente, aprovechando el fenómeno del entrelazamiento. Esto hace que se puedan realizar cálculos que serían imposibles en ordenadores clásicos o en su defecto, muy costosos. Algunos algoritmos que aprovechan y explotan todas estas ventajas pueden factorizar números grandes, buscar elementos en una lista desordenada o simular sistemas físicos complejos mucho más eficientemente que otros algoritmos clásicos. En esta sección se estudiarán algunos algoritmos básicos buscando el algoritmo de resolución de sistemas lineales, el cual otorga una mejora exponencialmente más rápida frente a su contraparte clásica, como se verá más adelante. Esto se traduce en aplicaciones en criptografía, inteligencia artificial y aprendizaje automático.

3.3.1. Transformada de Fourier Cuántica

La transformada de Fourier es una herramienta matemática fundamental en el análisis de señales y sistemas, utilizada para descomponer una señal en sus componentes de frecuencia. Fue desarrollada por el matemático francés Jean-Baptiste Joseph Fourier en el siglo XIX y ha demostrado ser una herramienta de suma importancia en diversas áreas como las matemáticas, la física, las ingenierías e incluso la ciencia de datos y aprendizaje automático.

Se utiliza para analizar señales y descomponerlas en una suma de componentes sinusoidales de diferentes frecuencias. Esto significa que cualquier señal, incluso señales no periódicas y complejas, puede ser expresada como una combinación de señales sinusoidales simples. Esta descomposición en frecuencias nos permite comprender mejor el contenido espectral de una señal y extraer información relevante sobre su comportamiento en el dominio de la frecuencia. La transformada de Fourier discreta está definida como

$$y_k \equiv \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{2\pi i j k/N} \tag{50}$$

Donde $\mathbf{x}$ son los datos de entrada, normalmente números complejos, y devuelve otro vector de números complejos $\mathbf{y}$.

La transformada de Fourier cuántica [6, 25] hace exactamente lo mismo, solo que con

estados cuánticos. Definida en una base ortonormal toma la siguiente forma:

$$|j\rangle \longrightarrow \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i j k/N} |k\rangle \tag{51}$$

De esta forma, en un estado arbitrario actuará como

$$\sum_{j=0}^{N-1} x_j |j\rangle \longrightarrow \sum_{k=0}^{N-1} y_k |k\rangle \tag{52}$$

donde y_k son las amplitudes obtenidas tras aplicar la transformada de Fourier discreta a las amplitudes x_i. Se puede probar que la transformada de Fourier cuántica es un operador unitario. Cabe destacar que tanto j como k son números binarios.

Para el desarrollo y la explicación de este algoritmo se supondrá que $N = 2^n$, con n el número de qubits. También se utilizará la notación binaria para enteros y decimales:

$$\begin{aligned} j_1 j_2 ... j_n =& j_1 \cdot 2^{n-1} + ... + j_n \cdot 2^0 \\ 0.j_l j_{l-1} ... j_{l-m+1} =& j_l \cdot 2^{-1} + ... + j_{l-m+1} \cdot 2^{-(m-l+1)} \end{aligned} \tag{53}$$

Partiendo de la transformada de j [6]:

$$|j\rangle \longrightarrow \frac{1}{2^{n/2}} \sum_{k=0}^{2^n-1} \epsilon^{2\pi i j k/2^n} |k\rangle \tag{54}$$

$$\frac{1}{2^{n/2}} \sum_{k_1=0}^{1} \cdots \sum_{k_n=0}^{1} \bigotimes_{l=1}^{n} e^{2\pi i j k_l 2^{-l}} |k_l\rangle \tag{55}$$

$$\frac{1}{2^{n/2}} \bigotimes_{l=1}^{n} \left(|0\rangle + e^{2\pi i j 2^{-l}} | \, |1\rangle \right) \tag{56}$$

Finalmente, si se expande el productorio tensorial:

$$\frac{\left(|0\rangle + e^{2\pi j 2^{-1}}|1\rangle\right)\left(|0\rangle + e^{2\pi j 2^{-2}}|1\rangle\right) \cdots \left(|0\rangle + e^{2\pi j 2^{-n}}|1\rangle\right)}{2^{n/2}} \tag{57}$$

El número $j/2^l$ se puede descomponer en su parte decimal y su parte entera en forma de suma. Sin embargo, como se esta tratando con un factor 2π delante, toda la exponencial con la parte entera vale 1, quedando tan solo la parte decimal.

$$\frac{\left(|0\rangle + e^{2\pi 0.j_n}|1\rangle\right)\left(|0\rangle + e^{2\pi 0.j_{n-1}j_n}|1\rangle\right) \cdots \left(|0\rangle + e^{2\pi 0.j_1 j_2 ... j_n}|1\rangle\right)}{2^{n/2}} \tag{58}$$

Esta última expresión hace que sea sencillo derivar un circuito para la transformada de Fourier cuántica. El primer producto no es más que una puerta Hadamard, debido a que en el caso que $j_n = 0$, se obtiene el estado $|+\rangle$ y en el caso que $j_n = 1$, se obtiene $(|0\rangle + e^{2\pi 0.j_n}|1\rangle) = \left(|0\rangle + e^{2\pi j_n/2}|1\rangle\right) = |-\rangle$. Generalizando para el resto de qubits, se llega a la puerta de rotación:

$$R_k = \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i/2^k} \end{pmatrix} \tag{59}$$

Así, la transformada de Fourier llevada a un circuito queda de la siguiente manera:

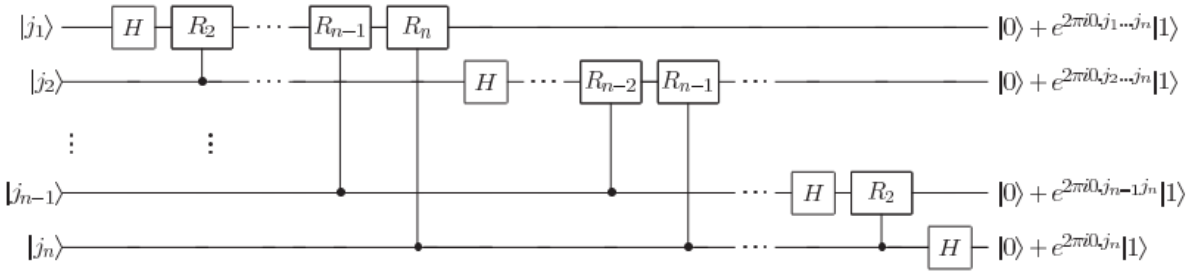


Figura 9: Circuito de la transformada de Fourier cuántica [25]

Esta operación se trata de un operador unitario, por lo que también se puede expresar en forma matricial tomando $\omega = e^{2\pi i/2^n}$:

$$QFT_N = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(N-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(N-1)} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \omega^{3(N-1)} & \cdots & \omega^{2(N-1)^2} \end{bmatrix} \tag{60}$$

Lo único que faltaría para cumplir con la definición dada al inicio de este apartado es aplicar puertas *SWAP* de forma que la última salida corresponda al primer qubit.

En cuanto a su complejidad computacional, se puede ver que en total se han aplicado $n(n+1)/2$ puertas, por tanto su complejidad será del orden de $O(n^2)$. En cuanto a su contraparte clásica, el mejor algoritmo encontrado se trata de *Fast Fourier Transformation* o por sus siglas *FFT*. Este tiene una complejidad del orden de $O(n2^n)$, por lo que comparándolo con el algoritmo cuántico, se ha conseguido obtener una mejoría exponencial.

Al parecer se podría considerar que se ha encontrado un algoritmo súper potente, con la capacidad de revolucionar muchas tareas que utilizan la transformada de Fourier. Sin embargo, a pesar de que el algoritmo cuántico sea exponencialmente más rápido que el clásico, hay unos cuantos inconvenientes. El primero de ellos es que las amplitudes de los estados no pueden ser directamente medidas. El siguiente y peor, es que no hay forma eficiente de preparar los estados originales a los que se les quiere aplicar la transformada de Fourier, por lo que parecería que encontrar alguna utilidad a este algoritmo es complicado. A pesar de esto, es una pieza clave en el desarrollo de algunos algoritmos utilizados para aplicar QSVMs, como se explorará a continuación.

3.3.2. Estimación de Fase

Se supone que se tiene un un operador U cuyos autovalores tienen la forma $e^{2\pi i\psi}$ y con autovectores $|u\rangle$.

$$|j\rangle U^k |u\rangle = |j\rangle e^{2\pi i\psi k} \tag{61}$$

El objetivo de la estimación de fase es el de obtener el valor de ψ, el cual se presupone desconocido. Para su aplicación, se asume que se tiene un conjunto de cajas negras (también llamadas oráculos) las cuales son capaces de generar el estado $|u\rangle$. La existencia de estas cajas negras implica que no se trata de un algoritmo completo, por lo que se tratará de subrutina.

La aplicación de esta subrutina tiene una precisión no infinita, la cual se puede mejorar aumentando el número de qubits utilizados. Para obtener el valor de ψ con una precisión de n bits, y con una probabilidad de éxito de la subrutina de $1 - \epsilon$, se deberá tener un número total de qubits t:

$$t = n + \left\lceil log\left(2 + \frac{1}{2\epsilon}\right)\right| \tag{62}$$

Para la aplicación de la estimación de fase, se deberán tener 2 registros de qubits. El primero será donde se almacene el valor de ψ, siendo un total de t qubits e inicializados todos a 0. El segundo registro es donde se inicialice el estado $|u\rangle$ y tendrá un total de qubits tan grande como se necesite para almacenarlo.

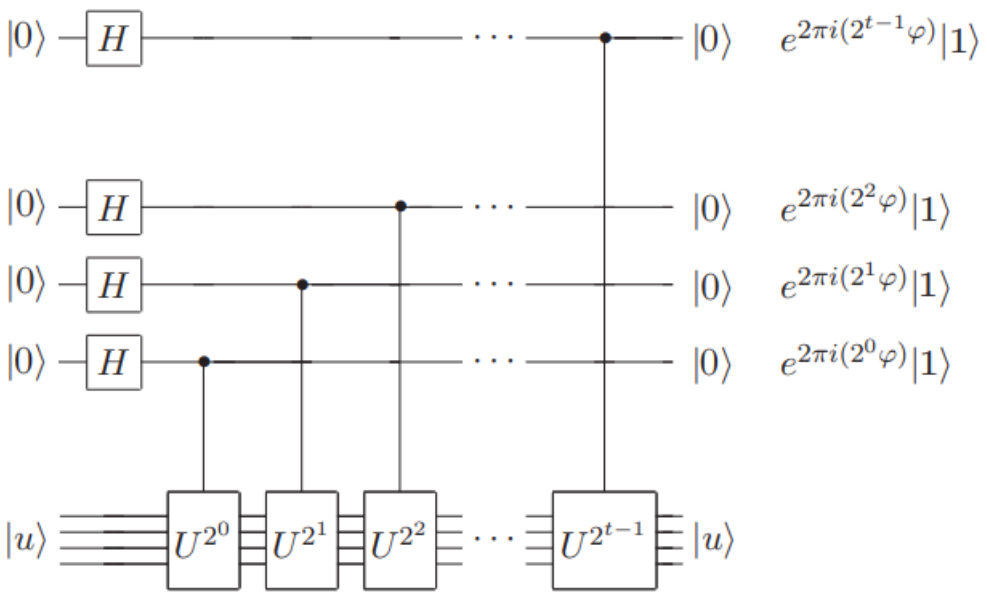


Figura 10: Puertas aplicadas en la subrutina de estimación de fase.

Aplicando los operadores unitarios U^{2^j} con $j = 0, 1 \cdots t-1$ en orden y controlados por los qubits del primer registro uno a uno tal y como se observa en la figura 10, se puede llegar a la siguiente expresión que representa el estado final:

$$\frac{1}{2^{t/2}}\left(|0\rangle + e^{2\pi i 2^{t-1}\varphi}|1\rangle\right)\left(|0\rangle + e^{2\pi i 2^{t-2}\varphi}|1\rangle\right)\cdots\left(|0\rangle + e^{2\pi i 2^{0}\varphi}|1\rangle\right) = \frac{1}{2^{t/2}}\sum_{k=0}^{2^t-1} e^{2\pi i\varphi k}|k\rangle \tag{63}$$

Una vez se tengan todos los estados preparados obteniendo así (63), se ha de aplicar la transformada de Fourier inversa. Esto es así, debido a que en realidad, habiendo preparado (63), es como si se hubiese obtenido un estado al que se le ha aplicado la transformada de Fourier cuántica. De esta forma, se podrán obtener los valores binarios de la fase cuántica que se buscaba.

$$FT^{\dagger}\left(\frac{1}{2^{t/2}}\sum_{j=0}^{2^t-1}\epsilon^{2\pi i\varphi j}|j\rangle|u\rangle\right) = |\tilde{\varphi}\rangle|U\rangle \tag{64}$$

En resumen, la subrutina para obtener la fase cuántica se puede escribir en forma de un circuito como el siguiente:

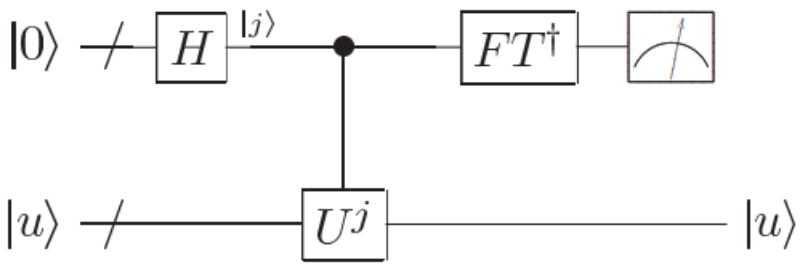


Figura 11: Esquemática general del circuito de la estimación de fase cuántica [25].

3.3.3. Resolución de sistemas lineales

El algoritmo cuántico de resolución de sistemas lineales fue desarrollado por Harrow, Hassidim y Lloyd en 2009 [15]. Este algoritmo es uno de las principales esperanzas de la computación cuántica debido a que, al igual que el algoritmo de Shor [32] o el de Grover [14], obtiene una mejora de la complejidad computacional exponencial en comparación a su contraparte clásica y además es de utilidad.

El problema parte de una matriz hermítica $N \times N$ A y un vector unitario $\mathbf{b}$, los cuales cumplen la relación $A\mathbf{x} = \mathbf{b}$, donde $\mathbf{x}$ es el vector incógnita.

Para empezar, primeramente se debe de expresar el vector unitario $\mathbf{b}$ en la base de los estados cuánticos:

$$|\mathbf{b}\rangle = \sum_{i=1}^{N} b_i |i\rangle \tag{65}$$

A continuación, se debe exponenciar la matriz A, de forma que se obtenga el operador unitario e^{iAt}, el cual si se aplica al estado $|\mathbf{b}\rangle$ y se aplica el algoritmo de estimación de fase cuántica se obtendrá el estado $|\mathbf{b}\rangle$ en la base de autovectores de A. Con esto se podrán obtener también los autovalores de la matriz A, dados por λ_j.

$$|\mathbf{b}\rangle = \sum_{j=1}^{N} \beta_j |u_j\rangle \tag{66}$$

$$e^{iAt} |\mathbf{b}\rangle \rightarrow \sum_{j=1}^{N} \beta_j |u_j\rangle |\lambda_j\rangle \tag{67}$$

A continuación se busca obtener la inversa de estos autovalores. Para ello, como la transformación lineal a aplicar es no unitaria, hay que iterar varias veces hasta que se consiga converger con un error definido.

$$|\lambda_j\rangle \to C \cdot \lambda_j^{-1} |\lambda_j\rangle \tag{68}$$

Con C una constante de normalización. De esta forma, se puede llegar a lo siguiente:

$$e^{iAt} |\mathbf{b}\rangle \to \sum_{j=1}^{N} \beta_j |u_j\rangle |\lambda_j\rangle \to \sum_{j=1}^{N} \beta_j \lambda_j^{-1} |u_j\rangle \tag{69}$$

Lo cual no sería nada menos que la inversa de la matriz A aplicada al estado $|\mathbf{b}\rangle$ y por tanto se ha obtenido la solución deseada:

$$\sum_{j=1}^{N} \beta_j \lambda_j^{-1} |u_j\rangle = A^{-1} |b\rangle = |\mathbf{x}\rangle \tag{70}$$

Donde $|\mathbf{x}\rangle$ es la representación mecanocuántica del vector inicial $\mathbf{x}$. Para poder obtener todas las componentes de este vector, sería necesario medirlo al menos N veces, es por ello que muchas veces es más útil extraer información de él mediante un operador M y obteniendo su valor medio. De esta forma se puede obtener información como los pesos.

La aplicación de este algoritmo para la resolución de sistemas lineales se ha demostrado tener una complejidad computacional de $O(\kappa^2 \log N/\epsilon)$, con ϵ es el parámetro de error y κ es el número de condición de A. Este último está relacionado con el ratio de los autovalores más grandes de A con los más pequeños.

En comparación, su contraparte clásica (por ejemplo, el algoritmo de eliminación de Gauss-Jordan) tiene una complejidad computacional es de $O(N^3)$. Por lo tanto, se ha conseguido una mejora exponencial con el algoritmo cuántico.

Este algoritmo ha sido aplicado repetidas veces de forma satisfactoria [4, 1, 27].

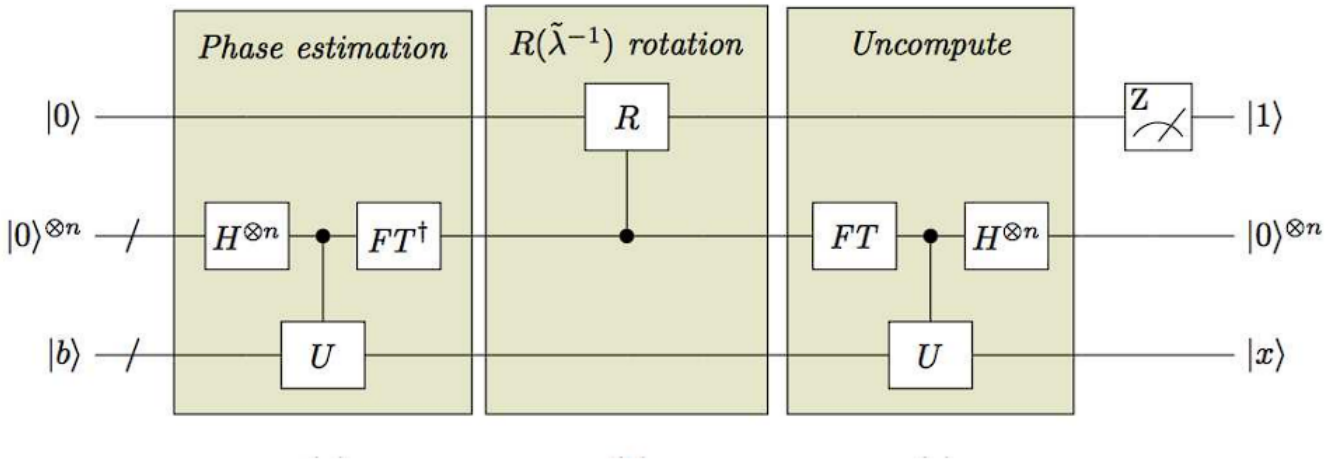


Figura 12: Descripción del algoritmo de resolución de sistemas lineales en un ordenador cuántico descrito en [10]. El primer qubit se trata del qubit ancilla o ayudante y será denotado como S. El segundo grupo de qubits será el registro y es denotado como C. El tercer grupo de qubits será nel *input*, siendo denotado como I.

En la figura 12 se puede observar el circuito cuántico desarrollado para la resolución de sistemas lineales. En el primer bloque, el cual se trata de obtener los autovalores de la matriz A sobre el vector **b**, el cual estará descrito en la base de autoestados de la matriz A.

El segundo bloque se trata de la rotación no unitaria para obtener los autovalores de A invertidos. Se llegará al siguiente estado:

$$\sum_{j=1}^{N} \beta_j \left|u_j\right\rangle^I \left|\tilde{\lambda}_j\right\rangle^C \left(\sqrt{1-\frac{C^2}{\tilde{\lambda}_j^2}}\left|0\right\rangle + \frac{C}{\tilde{\lambda}_j}\left|1\right\rangle\right)^S \tag{71}$$

Tras haber aplicado $\exp(-i\theta\sigma_y) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$ con $\theta = \cos^{-1}\left(\frac{C}{\tilde{\lambda}_j}\right)$, tomando C como una constante de desnormalización.

El tercer bloque se refiere a la descomputación, donde se obtiene el resultado buscado. Este devolverá el estado

$$\sum_{j=1}^{N} \beta_j \left|u_j\right\rangle^I \left(\sqrt{1-\frac{C^2}{\lambda_j^2}}\left|0\right\rangle + \frac{C}{\tilde{\lambda}_j}\left|1\right\rangle\right)^S \tag{72}$$

Quedándose con los estados $|1\rangle$ del qubit ancilla S, se obtendrá el estado final $C\sum_{j=1}^{N}(\frac{\beta_j}{\lambda_j})\left|u_j\right\rangle^I$.

La inversa de la matriz A se podrá escribir como $\sum_j \frac{1}{\lambda_j} |u_j\rangle \langle u_j\rangle$ y por tanto

$$A^{-1}\mathbf{b} \equiv \sum_{j=1}^{N} \frac{\beta_j}{\lambda_j} |u_j\rangle^I = |x\rangle \tag{73}$$

De esta forma, se ha conseguido alcanzar el estado deseado de la solución a partir de un sistema de ecuaciones lineales clásicas. Cabe destacar que el problema principal de este algoritmo es la correcta exponenciación de la matriz. Esto es un problema totalmente aparte, llamado 'Simulación del Hamilitoniano' el cual requiere que la matriz sea dispersa para poder realizarse de forma eficiente [3].

La resolución de sistemas de ecuaciones lineales en ordenadores cuánticos se ha llevado a cabo satisfactoriamente numerosas ocasiones [5, 1, 28], convirtiéndose en un algoritmo el cual tiene un enorme potencial.

4. QSVMs

Partiendo de lo que se ha visto en los apartados anteriores, parece difícil a simple vista tratar de construir en un ordenador cuántico una máquina de soporte vectorial. Sin embargo, no solo es totalmente posible, sino que además, se ha demostrado que se llega a una conclusión de forma hasta exponencialmente más rápida [23].

Las máquinas de vectores de soporte cuánticas (QSVMs) son una extensión de las máquinas de vectores de soporte (SVMs) clásicas que aprovechan las ventajas de la computación cuántica. Las QSVMs pueden resolver problemas de clasificación y regresión con una mayor eficiencia y precisión que las SVMs tradicionales, especialmente cuando los datos son de alta dimensión o no linealmente separables. Esto se debe a que por ejemplo, tienen la capacidad de utilizar estados cuánticos superpuestos y entrelazados para representar los datos y el kernel, lo que les permite acceder a un espacio de características más rico y complejo que las SVMs clásicas. Además, pueden aprovechar el paralelismo cuántico y la interferencia para realizar cálculos más rápidos y eficientes que las SVMs clásicas.

En los siguientes apartados se van a estudiar distintos tipos de algoritmos de QSVMs y sus principales diferencias entre ellos y su contraparte clásica.

4.1. Algoritmo Cuántico de Máquinas de Soporte Vectorial

Este algoritmo [31] parte de la 'linealización' dada por la formulación de las SVM mediante mínimos cuadrados, la cual reduce el problema cuadrático a un sistema de ecuaciones lineales.

Se parte un conjunto de datos de entrenamiento que serán oráculos $\mathbf{x}_j$, de los cuales se conoce su norma $\|\mathbf{x}_j\|$ y su etiqueta y_j. Esto resulta en un conjunto de vectores de entrenamiento los cuales se pueden representar en el ordenador cuántico como:

$$|\mathbf{x}_j\rangle \Longrightarrow \frac{1}{|\mathbf{x}_j|}\sum_{k=1}^{N}(\mathbf{x}_j)_k\,|k\rangle \tag{74}$$

Donde $j = 1, 2, ..., M$ es el número de datos de entrenamiento y N será el número de dimensiones del espacio de características.

A continuación se calculará la matriz del kernel, la cual juega un papel fundamental tanto en la formulación dual como en la formulación de mínimos cuadrados (ecuaciones (25), (33)). Esta, clásicamente se calcula tomando todos los datos de entrenamiento y

aplicando el kernel en parejas. Esto da como resultado una matriz semi-definida positiva. De forma cuántica, se hará parecido:

$$\frac{K}{Tr(K)} = \frac{1}{N_\chi} \sum_{i,j=1}^{M} \langle \mathbf{x}_j | \mathbf{x}_i \rangle |\mathbf{x}_i||\mathbf{x}_j| \, |i\rangle \langle j| \tag{75}$$

Con $N_\chi = \sum_{i=1}^{M} |\mathbf{x}_i|$.

A partir del cómputo de esta matriz se podrá llegar a la formulación de mínimos cuadrados vista en la ecuación (33), que recordándola:

$$F \cdot \begin{pmatrix} b \\ \boldsymbol{\alpha} \end{pmatrix} = \begin{pmatrix} 0 & \mathbf{1}^\top \\ \mathbf{1} & K + \gamma^{-1} I \end{pmatrix} \begin{pmatrix} b \\ \boldsymbol{\alpha} \end{pmatrix} = \begin{pmatrix} 0 \\ \mathbf{y} \end{pmatrix} \tag{76}$$

En esta ecuación matricial se pueden diferenciar varios elementos: los vectores $\mathbf{1} = (1, \ldots, 1)^T$, γ tendrá la misma función que C, la función coste, I será la matriz identidad y el vector $\begin{pmatrix} b \\ \boldsymbol{\alpha} \end{pmatrix}$ será la incógnita. La matriz F tendrá un tamaño de $(M+1) \times (M+1)$, donde M es la cantidad de datos de entrenamiento disponibles. La fila y columna extras corresponden a un posible valor no nulo de b. De esta forma, se busca conseguir la matriz F^{-1} de forma que se pueda obtener el vector incógnita $\begin{pmatrix} b \\ \boldsymbol{\alpha} \end{pmatrix} = F^{-1} \begin{pmatrix} 0 \\ \mathbf{y} \end{pmatrix}$. Este algoritmo, de forma clásica, está sujeto a un coste de $\mathrm{O}(M^3)$ (a falta de optimización con otros algoritmos más eficientes, pero no se consigue una reducción significativa).

De forma cuántica, la tarea de una QSVM es partir del sistema siguiente:

$$\hat{F} \, |b, \boldsymbol{\alpha}\rangle = |\mathbf{y}\rangle \tag{77}$$

Donde el estado cuántico $|b, \boldsymbol{\alpha}\rangle$ describe el hiperplano que separa el conjunto de datos y $\hat{F} = F/Tr\, F$, de forma que al igual que clásicamente, se llegue a:

$$|b, \boldsymbol{\alpha}\rangle = \hat{F}^{-1} \, |\mathbf{y}\rangle \tag{78}$$

Con esto, se podrán clasificar nuevos estados $|\mathbf{x}\rangle$, tal y como se hacía clásicamente.

Para la aplicación del algoritmo de inversión de matriz, se requiere que $\hat{F}$ sea exponenciada eficientemente. Es por ello que se recurre a separarla de la siguiente forma:

$$\hat{F} = (J + K + \gamma^{-1} I) / \operatorname{Tr} F \tag{79}$$

donde $J = \begin{pmatrix} 0 & \mathbf{1}^T \\ \mathbf{1} & 0 \end{pmatrix}$ y de esta forma se puede descomponer la exponenciación de $\hat{F}$ haciendo uso del producto de Lie de la siguiente forma:

$$e^{-i\hat{F}\Delta t} = e^{-i\Delta t I/\mathrm{Tr}F} e^{-iJ\Delta t/\mathrm{Tr}F} e^{-iK\Delta t/\mathrm{Tr}F} + O(\Delta t^2) \tag{80}$$

Para calcular la inversa de las matrices, se debe de conseguir la exponencial de estas matrices de forma eficiente, llegando a un problema de simulación de hamiltonianos. Para ello, es obligatorio que la matriz sea dispersa [2]. Primeramente se deberán conseguir los autovalores de las matrices. Los autovalores de la matriz J son directos y vendrán dados por $\lambda_{\pm}^{*} = \pm\sqrt{M}$, con los autoestados $|\lambda_{\pm}^{*}\rangle = 1/\sqrt{2}\left(|0\rangle \pm 1/\sqrt{M}\sum_{k=1}^{M}|k\rangle\right)$. Los de $\gamma^{-1} \times I$ son triviales por tratarse de una matriz ya diagonal. En cuanto la matriz K, se ha de recurrir a un algoritmo cuántico más complejo dado por los mismos autores [24] que encuentre las componentes principales ya que se trata de una matriz no dispersa. Con todo esto, se puede obtener $e^{-i\hat{F}\Delta t}$.

En cuanto a $|\mathbf{y}\rangle$, se puede expresar en la base de autoestados $|u_j\rangle$ de $\hat{F}$ de forma que:

$$|\tilde{y}\rangle \;=\; \sum_{j=1}^{M+1} \langle u_j|\mathbf{y}\rangle |u_j\rangle \tag{81}$$

Mediante un registro para guardar la aproximación de los autovalores, se llega a un estado el cual si se invierte realizando una inversión controlada:

$$|\tilde{y}\rangle|0\rangle \rightarrow \sum_{j=1}^{M+1} \langle u_j|\tilde{y}\rangle |u_j\rangle\,|\lambda_j\rangle \rightarrow \sum_{j=1}^{M+1} \frac{\langle u_j|\tilde{y}\rangle}{\lambda_j}|u_j\rangle \tag{82}$$

La expansión de los coeficientes del nuevo estado se trata de los parámetros que describen la QSVM:

$$|b, \boldsymbol{\alpha}\rangle = \frac{1}{\sqrt{C}}\left(b|0\rangle + \sum_{k=1}^{M} \alpha_k |k\rangle\right) \tag{83}$$

Una vez calculados los coeficientes de la QSVM, y por tanto, habiéndola entrenado, se puede pasar a la clasificación de datos nuevos los cuales se denotarán como $|\mathbf{x}\rangle$. Para ello, se construirá el oráculo de los datos de entrenamiento siguiente:

$$|\tilde{u}\rangle = \frac{1}{\sqrt{N_{\tilde{u}}}}\left(b|0\rangle|0\rangle + \sum_{k=1}^{M} \alpha_k |\mathbf{x}_k||k\rangle|\mathbf{x}_k\rangle\right) \tag{84}$$

Con $N_{\tilde{u}} \;=\; b^2 + \sum_{k=1}^{M} \alpha_k^2 |\mathbf{x}_k|^2$.

Por otra parte, se construirá la 'consulta' siguiente:

$$|\tilde{x}\rangle = \frac{1}{\sqrt{N_{\tilde{x}}}} \left(|0\rangle|0\rangle + \sum_{k=1}^{M} |\mathbf{x}||k\rangle|\mathbf{x}\rangle \right) \tag{85}$$

Con $N_{\mathbf{x}} = M|\mathbf{x}|^2 + 1$. Ahora, se construirá el estado de soporte $|\psi\rangle = \frac{1}{\sqrt{2}}\Big(|0\rangle\,|\tilde{u}\rangle + |\,1\rangle\,|\tilde{x}\,\rangle \Big)$ y se procede a medir este estado sobre $|\phi\rangle = \frac{1}{\sqrt{2}}\,(|0\rangle - |1\rangle)$. La probabilidad de que la medición sea correcta es de $P = |\langle\psi|\phi\rangle|^2 = \frac{1}{2}(1 - \langle\tilde{u}|\tilde{x}\rangle)$, con el producto escalar dado por:

$$\langle\tilde{u}|\tilde{x}\rangle = \frac{1}{\sqrt{N_{\mathbf{x}} N_{\mathbf{u}}}} \left(b + \sum_{k=1}^{M} \alpha_k |\mathbf{x}_k||\mathbf{x}|\langle\mathbf{x}_k|\mathbf{x}\rangle \right) \tag{86}$$

Un P correcto puede ser encontrado con un error ϵ iterando $O(P(1-P)/\epsilon^2)$. De esta forma, se puede decir que si $P < 1/2$, $y(\mathbf{x}) = +1$. De lo contrario, será -1. De esta forma, se consigue crear un clasificador que escala en tiempos con $O(\log MN)$.

Cabe destacar que para la aplicación de este algoritmo también es posible aplicar funciones de incrustación a los datos de entrenamiento. De esta forma se conseguirá un mapeado no lineal en un espacio vectorial que puede tener más dimensiones que el original. Un ejemplo de estos mapeados no lineales es el kernel polinomial $k(\mathbf{x}_i, \mathbf{x}_k) = (\mathbf{x}_i \cdot \mathbf{x}_k)^d$. Para la aplicación de este kernel, se utilizará la función de incrustación siguiente:

$$|\phi(\mathbf{x}_j)\rangle = |\mathbf{x}_j\rangle \otimes |\mathbf{x}_j\rangle \otimes .... |\mathbf{x}_j\rangle \tag{87}$$

De esta forma, $\langle\phi(\mathbf{x}_j)|\phi(\mathbf{x}_k)\rangle = \langle\mathbf{x}_j\,|\mathbf{x}_k\rangle^d$, convirtiendo el kernel polinomial original en un kernel lineal calculado d veces.

Con todo esto, se ha estudiado que es completamente posible la construcción de un clasificador binario en un ordenador cuántico. Sin embargo, está apoyado sobre una base muy grande y aún en estudio de algoritmia cuántica, como podría ser el problema de la simulación de hamiltonianos. A pesar de esto, es un algoritmo completamente funcional, aunque complicado de aplicar.

4.1.1. Aplicación del Algoritmo

En esta sección se estudiará la aplicación del algoritmo desarrollado por Lloyd, Rebentrost y Mohseni y explicado en el apartado anterior. Para esto, se acudirá al experimento realizado por Zhaokai Li, Xiaomei Liu, Nanyang Xu y Jiangfeng Du [22] el cual fue realizado en un ordenador cuántico de 4 qubits de resonancia magnética nuclear.

El problema a resolver se trata de una imagen donde se busca diferenciar dos dígitos manuscritos, el 6 y el 9. Cada vector de datos contendrá 2 características. Las características serán las siguientes: el ratio de píxeles verticales, donde se comparan la cantidad de píxeles negros de la mitad arriba y la mitad abajo; y el ratio de píxeles horizontales, donde se comparan la mitad derecha con la izquierda. Los datos de entrenamiento para este caso serán imágenes de los dígitos 6 y 9 en una fuente estándar. A estos datos se les ha aplicado una conversión lineal y se han normalizado de forma que quepan en el procesador de 4 qubits.

Training data (printed characters)		label
6	$\vec{x}_1 = (0.987, 0.159)$	$y(\vec{x}_1) = +1$
9	$\vec{x}_2 = (0.354, 0.935)$	$y(\vec{x}_2) = -1$

(a) Datos de entrenamiento en una fuente estándar

Handwritten characters			
6	(0.997, -0.072)	9	(0.338, 0.941)
9	(0.147, 0.989)	6	(0.999, 0.025)
6	(0.999, -0.030)	9	(0.439, 0.899)
6	(0.987, -0.161)	9	(0.173, 0.985)

(b) Datos del escaneo de dígitos manuscritos

Figura 13: Datos obtenidos por los autores [22]

Una vez se han definido los vectores de entrenamiento, se les debe de dar una etiqueta. Cuando el número sea identificado como un 6, se devolverá $y(6) = 1$, y si es un 9 $y(9) = -1$

En cuanto a la parte experimental, los autores procedieron mediante un espectrómetro Bruker AV-400 a 306K. La muestra utilizada se trata de C_2F_3I disuelto en *d*-cloroformo, obteniendo un qubit por parte del ^{13}C y 3 dados por el ^{19}F.

Para el cómputo de la matriz $\tilde{K}$, se procede a aplicar el circuito de la figura 14

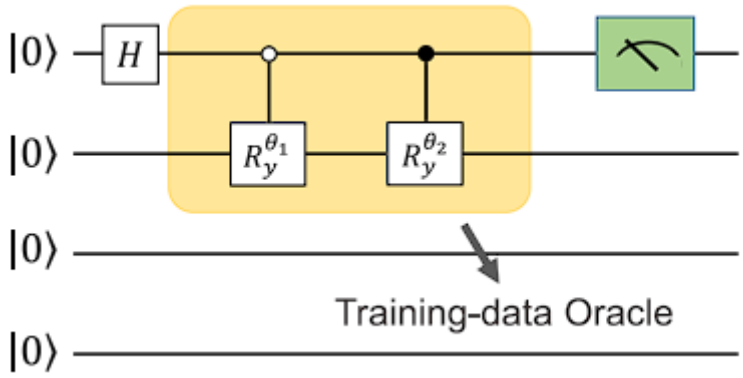


Figura 14: Calculo de la matriz del kernel como una matriz de densidad [22]

En él, se utiliza el segundo qubit de modo que se codifica la información de los datos de entrenamiento en él. Para ello, se utilizan dos puertas de rotación cuya fase vendrá dada por $\theta_i = arccot(\frac{(x_i)_1}{(x_i)_2})$. La matriz obtenida fue la siguiente:

$$\frac{K}{TrK} = \begin{bmatrix} 0{,}5065 & 0{,}2425 \\ 0{,}2425 & 0{,}4935 \end{bmatrix} \tag{88}$$

A continuación se procede con la clasificación de datos nuevos. El circuito a implementar fue el siguiente:

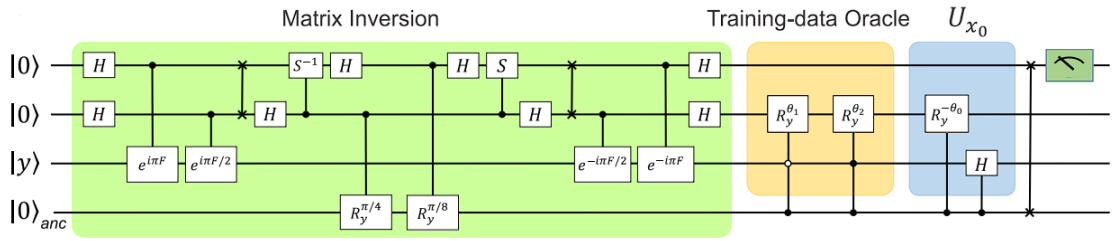


Figura 15: Circuito de la clasificación, donde H será la puerta Hadamard y S será la puerta de fase [22]

Es la figura 15 se pueden observar los 3 algoritmos principales. El primero de ello es la inversión de la matriz F descrita en el algoritmo. Sin embargo, cabe destacar que en este

caso, como se tomará $b = 0$, y por tanto el sistema a resolver será $F \cdot \boldsymbol{\alpha} \equiv (K + \gamma^{-1}\mathbf{I})\boldsymbol{\alpha} = \mathbf{y}$ con $\gamma = 2$ un peso arbitrario. Los dos primeros qubits actúan como un registro para realizar la inversión de la matriz y por tanto generando el estado $F^{-1}|y\rangle$ en el tercer qubit. Con esto, los valores α_i ya han sido calculados y almacenados en el registro. Llamando de nuevo al oráculo de los datos de entrenamiento, se procede a construir el estado $|\tilde{u}\rangle$, de forma que se puede dar comienzo a la medición.

A diferencia que en algoritmo presentado [31], para la medición en este experimento se desarrolla el operador unitario U_{x_0}, descrito en la figura 15, el cual prepara el estado $|x_0\rangle$ de forma que al hacer el producto escalar sobre $|u\rangle$ se obtenga el resultado de la clasificación. Este producto escalar dará un número positivo o negativo. Si es positivo se le dará la etiqueta $y = +1$. Si es negativo $y = -1$

Los resultados de la clasificación se mostrarán en el primer qubit, hecho por el carbono. Si el pico correspondiente al espectro del carbono está para arriba, la clasificación será positiva, dando a entender que el reconocimiento del carácter da como resultado un 6. Si por el contrario sale para abajo, el resultado del reconocimiento será un 9.

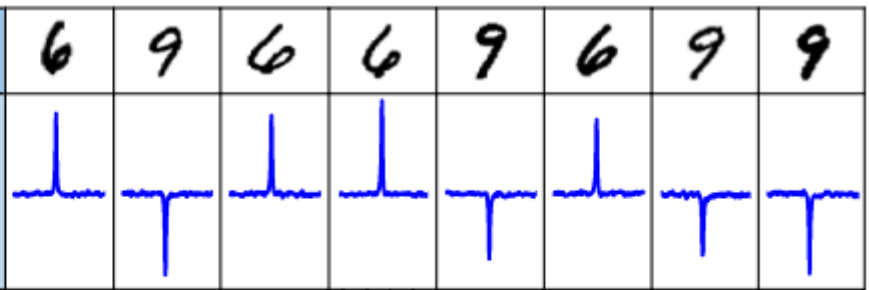
Hand-written characters								
Experimental indicators								
Amplitude	0.2234	-0.2247	0.2205	0.2496	-0.1775	0.2092	-0.1421	-0.2278
Recognition results	6	9	6	6	9	6	9	9

Figura 16: Resultados del reconocimiento de caracteres escritos a mano. Las filas representan por orden: los caracteres dibujados a mano, los indicadores experimentales, la amplitud del término coherente y los resultados [20].

Como se ha podido ver, se ha conseguido obtener una máquina de soporte vectorial cuántica capaz de clasificar elementos procesados de forma óptica, los cuales se han llevado a un ordenador cuántico, consiguiendo aplicar el clasificador con un coste computacional exponencialmente más bajo a una clásica.

4.2. Máquina de Soporte Vectorial Mejorada Cuánticamente

Hasta ahora se han estudiado máquinas de soporte vectorial puras, tanto puramente clásicas como puramente cuánticas. Sin embargo, en un ordenador cuántico se pueden conseguir algunos kernels que de forma clásica son imposibles de reproducir o si el espacio de características es muy grande la aplicación de estos se vuelve totalmente ineficiente. Por otra parte, en el método explicado en la sección anterior, los datos no pueden ser dados de una forma clásica con un ordenador convencional de forma eficiente. Es por ello que en el artículo de Havlíček [16] se da un algoritmo que utiliza máquinas de soporte vectorial clásicas para la tarea de clasificación y a su vez espacios de características cuánticos para obtener una mejora y un enriquecimiento del espacio de características respecto a un SVM clásico.

Al igual que en los casos anteriores, se parte de un conjunto de datos T (datos de entrenamiento) y S (datos de test). Para sendos conjuntos de datos se tiene las etiquetas de cada uno de ellos $y_k = \{-1, 1\}$, pero el algoritmo tan solo conocerá las de T. El objetivo es conseguir que $y(S) = \tilde{m}$ sea lo más parecido posible al original.

El requisito principal para obtener una mejora respecto al SVM clásico es que el kernel no se pueda simular eficientemente. Esto se puede aprovechar mediante la computación cuántica debido a que la simulación de un circuito cuántico de forma clásica es muy costosa, mientras que su aplicación en un ordenador cuántico real a penas tiene coste computacional. Es cierto que un kernel demasiado simple puede ser simulado de forma muy rápida, es por ello que se deben complicar los circuitos de forma que se obtenga la mejora cuántica. En este artículo se propuso el siguiente operador unitario:

$$\mathcal{U}_\Phi(\mathbf{x}) \;=\; U_{\Phi(\mathbf{x})} H^{\otimes n} U_{\Phi(\mathbf{x})} H^{\otimes n} \tag{89}$$

Donde H denota a la puerta Hadamard y el operador $U_{\Phi(\mathbf{x})}$ se define como

$$U_{\Phi(\mathbf{x})} = \exp\left(i \sum_{S \subseteq [n]} \phi_S(\mathbf{x}) \prod_{i \in S} Z_i \right), \tag{90}$$

Donde S indica las interacciones entre los qubits. De esta forma, para un $S \leq 2$, solo se considerará que como mucho interactuarán de dos en dos. Esto significa que el producto $\prod_{i \in S} Z_i$ llevará a como mucho interacciones $Z_i Z_j$ y términos no cruzados Z_i. Se definen las funciones clásicas $\phi_i(\mathbf{x}) = \mathbf{x}$. $\phi_{i,j}(\mathbf{x}) = (\pi - \mathbf{x}_i)(\pi - \mathbf{x}_j)$, aunque para distintos conjuntos de datos se pueden definir distintas funciones clásicas. Un ejemplo para dos qubits y $S \leq 2$, vendría dado por la siguiente expresión:

$$U_{\Phi(x)} = \exp\left(i \left(x_1 Z_1 + x_2 Z_2 + (\pi - x_1)(\pi - x_2) Z_1 Z_2 \right) \right) \tag{91}$$

Su implementación en un circuito de forma general se puede observar en la figura 17.

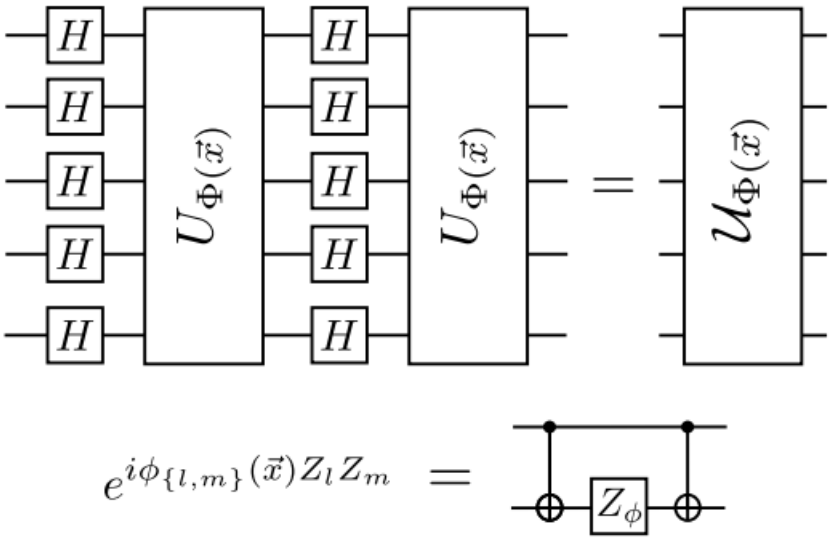


Figura 17: Circuito utilizado para la simulación de $\mathcal{U}_{\Phi(\mathbf{x})}$ [16]

Este algoritmo utiliza una máquina de vectores de soporte convencional, por lo que cabe recordar el problema de optimización de las SVMs clásicas.

$$\mathcal{L}_D(\boldsymbol{\alpha}) = \sum_{i=1}^{t} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{t} y_i y_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) \tag{92}$$

será el lagrangiano a optimizar. Este estará sujeto a las ligaduras $\sum_{i=1}^{t} \alpha_i y_i = 0$ y $\alpha_k \geq 0$ para todo k. La solución a este problema da como resultado un vector $\boldsymbol{\alpha} = (\alpha_1, \alpha_2 \cdots \alpha_t)$. De esta forma, se construye un clasificador:

$$\tilde{m}(\mathbf{s}) = \text{sign}\left(\sum_{i=1}^{t} y_i \alpha_i K(\mathbf{x}_i, \mathbf{s}) + b \right) \tag{93}$$

Tal y como se hizo en las primeras secciones de este trabajo.

Para poder obtener el kernel cuántico, se seguirá el siguiente protocolo. Primero, para cada par de datos se deberán preparar las puertas $U_{\Phi(\mathbf{x})}$, de forma que se tengan preparados los circuitos cuánticos $\mathcal{U}_\Phi(\mathbf{x})$. Con esto ahora se busca aplicar el producto escalar entre los dos pares de la siguiente forma:

$$|\langle \Phi(\mathbf{x}) \mid \Phi(\mathbf{z}) |^2 = |\langle 0^n | U_\Phi^\dagger(\bar{x}) \bar{U}_\Phi(\bar{z}) | 0^n \rangle|^2 \tag{94}$$

Esto se traducirá en aplicar el siguiente circuito cuántico:

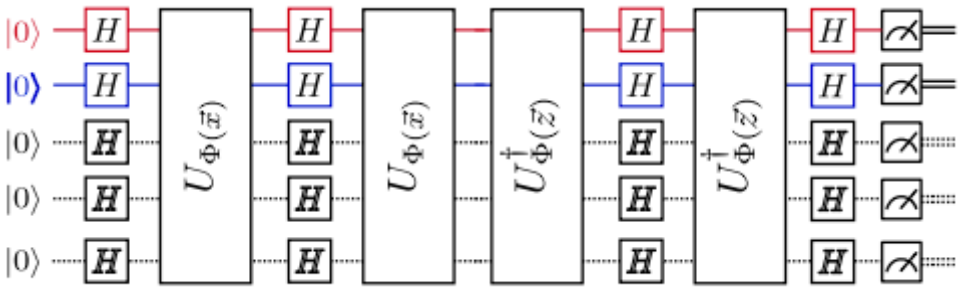


Figura 18: Circuito cuántico encargado de hacer el producto escalar entre los dos vectores en el espacio de características cuántico [16]

Una vez aplicado a los estados iniciales $|0\rangle^n$, se procederá a medir. Se contarán todos los ceros y este proceso se repetirá unas R veces. La frecuencia medida de estos ceros para cada par de datos será el valor de la componente correspondiente de la matriz kernel. Con esto, se obtendrá una precisión de $\|K - \hat{K}\| \leq \epsilon$, con $R \sim O(\epsilon^{-2})$. El coste del entrenamiento será de $O(T^2)$ con T el número de vectores de entrenamiento, que será el número de amplitudes a estimar. Por tanto, el coste total será de $O(\epsilon^{-2}T^2)$.

Una puerta $U_{\phi(\mathbf{x})}$ más general se puede conseguir con la siguiente expresión:

$$U_{\phi(x)} = \exp(i \sum_{j=1}^{n} \xi_j \phi_s(x) \prod \sigma_{j\in\{X,Y,Z\}}) \tag{95}$$

Donde σ_j representa las matrices de Pauli y ξ_j son factores de rotación [29].

Con esto se ha demostrado que se es capaz de construir una matriz kernel a partir de un circuito cuántico que no es eficiente simular clásicamente. Este nuevo kernel cuántico puede construir espacios de características nuevos y más ricos, aportando una ventaja sustancial a las máquinas de soporte vectorial clásicas, innovando en el sector de una forma única y dándoles una nueva aplicación a los ordenadores cuánticos.

En cuanto cómo aplicarlo, hay funciones en Qiskit [30] que pueden construir el circuito directamente. Es por ello que se ha desarrollado en el apéndice B, en el cual además se ha hecho una pequeña introducción a Qiskit. Se pueden consultar distintos artículos donde se ha aplicado satisfactoriamente [7], [16].

5. Conclusiones

En este trabajo de final de grado, cuyo propósito es meramente bibliográfico, se ha estudiado un algoritmo de aprendizaje automático y se ha llevado a ordenadores cuánticos. Se ha estudiado que se puede llegar a obtener una mejora exponencial en los costes computacionales o que se pueden obtener espacios de características mucho más ricos que pueden aportar ventajas en la clasificación.

Para ello, se han estudiado las máquinas de soporte vectorial viendo que se tratan de clasificadores binarios los cuales se utilizan un hiperplano para separar las dos clases dadas. Estas resuelven un problema de optimización mediante el cuál se llegan a las componentes del hiperplano. También se ha visto que se puede expresar como un problema de sistemas de ecuaciones lineales. Además, se ha estudiado cómo se puede mapear de forma no lineal los datos de entrenamiento, consiguiendo que el hiperplano que separa ambas características sea no lineal también.

A continuación, se ha introducido la computación cuántica, explicando cuales son sus bases y algunos algoritmos que se han utilizado. Se ha estudiado el algoritmo de resolución de sistemas de ecuaciones lineales, concluyéndose que es exponencialmente más eficiente que los algoritmos clásicos y por tanto abriendo la puerta para el aprendizaje automático cuántico.

Finalmente, se han visto dos algoritmos de máquinas de soporte vectorial cuánticas que aprovechan las cualidades de los ordenadores cuánticos para aportar una mejora computacional o para enriquecer y mejorar el mapeo de datos de las máquinas de soporte vectorial clásicas. En la primera, puramente cuántica, se aprovecha el algoritmo de resolución de sistemas de ecuaciones lineales para resolver el problema de máquinas de soporte vectorial en mínimos cuadrados. En la segunda, se construye un kernel cuántico el cual no se podría obtener de forma meramente clásica. Visto esto, se puede decir que los ordenadores cuánticos tienen un potencial todavía por explotar, y que cuando se consiga, van a traer una revolución mucho más grande de lo que cabría esperarse.

En conclusión, se puede decir que se han cumplido los objetivos, y que las máquinas de soporte vectorial cuánticas pueden llegar a ser muy útiles en un futuro donde la tecnología de los ordenadores cuánticos esté más desarrollada y sea más accesible para todos.

Referencias

[1] Stefanie Barz, Ivan Kassal, Martin Ringbauer, Yannick Ole Lipp, Borivoje Dakić, Alán Aspuru-Guzik, and Philip Walther. A two-qubit photonic quantum processor and its application to solving systems of linear equations. *Scientific Reports*, 4(1), August 2014.

[2] Dominic W. Berry, Graeme Ahokas, Richard Cleve, and Barry C. Sanders. Efficient quantum algorithms for simulating sparse hamiltonians. *Communications in Mathematical Physics*, 270(2):359–371, December 2006.

[3] Dominic W. Berry, Andrew M. Childs, Richard Cleve, Robin Kothari, and Rolando D. Somma. Exponential improvement in precision for simulating sparse hamiltonians. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*. ACM, May 2014.

[4] X.-D. Cai, C. Weedbrook, Z.-E. Su, M.-C. Chen, Mile Gu, M.-J. Zhu, Li Li, Nai-Le Liu, Chao-Yang Lu, and Jian-Wei Pan. Experimental quantum computing to solve systems of linear equations. *Physical Review Letters*, 110(23), June 2013.

[5] X.-D. Cai, C. Weedbrook, Z.-E. Su, M.-C. Chen, Mile Gu, M.-J. Zhu, Li Li, Nai-Le Liu, Chao-Yang Lu, and Jian-Wei Pan. Experimental quantum computing to solve systems of linear equations. *Physical Review Letters*, 110(23), June 2013.

[6] Jorge Calera Rubio. *Apuntes de computación cuántica para la asignatura de física cuántica avanzada*. Universitat d'Alacant, 2023.

[7] Bang-Shien Chen and Jann-Long Chern. Generating quantum feature maps for svm classifier, 2022.

[8] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.

[9] Joseph de Brabanter, Bart De Moor, Johan A K Suykens, and Joos Vandewalle. *Least Squares Support Vector Machines*. World Scientific Publishing, Singapore, Singapore, November 2002.

[10] Sanchayan Dutta, Adrien Suau, Sagnik Dutta, Suvadeep Roy, Bikash K. Behera, and Prasanta K. Panigrahi. Quantum circuit design methodology for multiple linear regression. *IET Quantum Communication*, 1(2):55–61, November 2020.

[11] A. Einstein, B. Podolsky, and N. Rosen. Can quantum-mechanical description of physical reality be considered complete? *Phys. Rev.*, 47:777–780, May 1935.

[12] Roberto González, Antonio Barrientos, Marcelo Toapanta, and Jaime del Cerro. Aplicación de las máquinas de soporte vectorial (svm) al diagnóstico clínico de la enfermedad de párkinson y el temblor esencial. *Revista Iberoamericana de Automática e Informática Industrial RIAI*, 14(4):394–405, 2017.

[13] James Manyika Google. An overview of bard: an early experiment with generative ai, 2023.

[14] Lov K. Grover. A fast quantum mechanical algorithm for database search, 1996.

[15] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Phys. Rev. Lett.*, 103:150502, Oct 2009.

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

[17] Laurence D Hoffman, Gerald L Bradley, and Laurence D Hoffmann. *Calculus for business, economics and the social and life sciences*. McGraw-Hill Education (ISE Editions), London, England, 8 edition, September 2003.

[18] Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425, 2002.

[19] IBM. IBM Quantum. https://quantum-computing.ibm.com/, 2021.

[20] Anekait Kariya and Bikash K. Behera. Investigation of quantum support vector machine for classification in nisq era, 2021.

[21] Yaoyong Li, Kalina Bontcheva, and Hamish Cunningham. Adapting svm for natural language learning: A case study involving information extraction, 2006.

[22] Zhaokai Li, Xiaomei Liu, Nanyang Xu, and Jiangfeng Du. Experimental realization of a quantum support vector machine. *Phys. Rev. Lett.*, 114:140504, Apr 2015.

[23] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum algorithms for supervised and unsupervised machine learning, 2013.

[24] Seth Lloyd, Masoud Mohseni, and Patrick Rebentrost. Quantum principal component analysis. *Nature Physics*, 10(9):631–633, July 2014.

[25] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, USA, 10th edition, 2011.

[26] OpenAI. Gpt-4 technical report, 2023.

[27] Jian Pan, Yudong Cao, Xiwei Yao, Zhaokai Li, Chenyong Ju, Hongwei Chen, Xinhua Peng, Sabre Kais, and Jiangfeng Du. Experimental realization of quantum algorithm for solving linear systems of equations. *Physical Review A*, 89(2), February 2014.

[28] Jian Pan, Yudong Cao, Xiwei Yao, Zhaokai Li, Chenyong Ju, Hongwei Chen, Xinhua Peng, Sabre Kais, and Jiangfeng Du. Experimental realization of quantum algorithm for solving linear systems of equations. *Physical Review A*, 89(2), February 2014.

[29] Jae-Eun Park, Brian Quanz, Steve Wood, Heather Higgins, and Ray Harishankar. Practical application improvement to quantum svm: theory to practice, 2020.

[30] Qiskit contributors. Qiskit: An open-source framework for quantum computing, 2023.

[31] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. Quantum support vector machine for big data classification. *Phys. Rev. Lett.*, 113:130503, Sep 2014.

[32] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, October 1997.

[33] Alex J. Smola, Bernhard Schölkopf, and Klaus-Robert Müller. The connection between regularization operators and support vector kernels. *Neural Networks*, 11(4):637–649, 1998.

[34] J. A. K. Suykens and J. Vandewalle. Least squares support vector machine classifiers. *Neural Processing Letters*, 9(3):293–300, Jun 1999.

[35] Peter Wittek. *Quantum Machine Learning: What Quantum Computing Means to Data Mining*. Academic Press, 08 2014.

A. Introducción a los Multiplicadores de Lagrange

A resumidas cuentas, los multiplicadores de Lagrange es un método matemático de optimización desarrollado por Joseph Louis Lagrange para obtener máximos o mínimos relativos en una función de varias variables sujetas a restricciones o ligaduras. Este método reduce el problema de n variables y k restricciones a un problema con $n+k$ variables sin restricciones [17].

Sea $f(\mathbf{x})$ una función definida en un conjunto abierto n-dimensional $x \in \mathbb{R}^n$, sujeta a las ligaduras $g_k(\mathbf{x}) = 0$ con $k = 1, ...m$ y $f, g \in \mathbf{C}^1$. Se define $h(\mathbf{x}, \boldsymbol{\lambda}) = f + \sum_{k=1}^{n} \lambda_k g_k$ como el lagrangiano.

Se buscan los extremos de $h(\mathbf{x}, \boldsymbol{\lambda})$, tomando $\frac{\partial h}{\partial \mathbf{x}_i} =$ que equivalentemente se reduce a:

$$\frac{\partial f}{\partial x_i} = -\sum_{k}^{n} \lambda_k \frac{\partial g_k}{\partial x_i}$$

Resolviendo el sistema de ecuaciones dado, se puede optimizar la función deseada.

B. Introducción a Qiskit y su QSVC

Qiskit [30] es un framework de código abierto desarrollado por IBM para trabajar con computación cuántica. Permite crear, simular y ejecutar programas cuánticos en dispositivos reales o simulados. Está escrito principalmente en python y proporciona una interfaz fácil de usar para interactuar con sistemas cuánticos. Para su correcto desarrollo, se recomienda hacer uso del laboratorio en IBMq [19].

Para crear un circuito cuántico en qiskit, primeramente se deberán de inicializar los qubits que se vayan a utilizar:

```
import qiskit as qk
# Creación de un registro cuántico
q = qk.QuantumRegister(2)
# Registro Clásico
c = qk.ClassicalRegister(2)
```

Para representar el circuito, se utilizará la función print:

```
circuito = qk.QuantumCircuit(q, c)
print(circuito)
```

A continuación, si lo que se busca es añadir puertas, se agregarán de la siguiente forma:

```
# Hadamard en el primer qubit
circuito.h(q[0])
# CNOT aplicado a el segundo qubit como precursor el primero
circuito.cx(q[0], q[1])
# Medición de todo el circuito sobre el registro clásico
circuito.measure(q, c)
print(circuito)
```

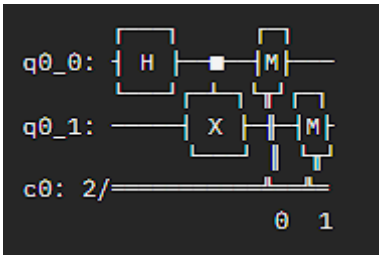


Figura 19: Circuito construido mediante qiskit

Para la aplicación del algoritmo de SVM's mejorados cuánticamente en qiskit, se seguirán los siguientes pasos, desarrollados con mucho más detalle en la documentación oficial de qiskit aquí [30]:

Primeramente, se importarán los datos utilizados en el artículo [16]:

```
from qiskit_machine_learning.datasets import ad_hoc_data
adhoc_dimension = 2
train_features, train_labels, test_features, test_labels, adhoc_total =
    ad_hoc_data(
     training_size=20,
     test_size=5,
     n=adhoc_dimension,
     gap=0.3,
     plot_data=False,
     one_hot=False,
     include_sample_total=True,
)
```

A continuación, se definen las funciones para representar el espacio de características.

```
import matplotlib.pyplot as plt
import numpy as np


def plot_features(ax, features, labels, class_label, marker, face, edge,
    label):
    # A train plot
    ax.scatter(
        # x coordinate of labels where class is class_label
        features[np.where(labels[:] == class_label), 0],
        # y coordinate of labels where class is class_label
```

```
        features[np.where(labels[:] == class_label), 1],
        marker=marker,
        facecolors=face,
        edgecolors=edge,
        label=label,
    )


def plot_dataset(train_features, train_labels, test_features, test_labels,
    adhoc_total):

    plt.figure(figsize=(5, 5))
    plt.ylim(0, 2 * np.pi)
    plt.xlim(0, 2 * np.pi)
    plt.imshow(
        np.asmatrix(adhoc_total).T,
        interpolation="nearest",
        origin="lower",
        cmap="RdBu",
        extent=[0, 2 * np.pi, 0, 2 * np.pi],
    )

    # A train plot
    plot_features(plt, train_features, train_labels, 0, "s", "w", "b", "A
    train")
    # B train plot
    plot_features(plt, train_features, train_labels, 1, "o", "w", "r", "B
    train")
    # A test plot
    plot_features(plt, test_features, test_labels, 0, "s", "b", "w", "A
    test")
    # B test plot
    plot_features(plt, test_features, test_labels, 1, "o", "r", "w", "B
    test")
    plt.legend(bbox_to_anchor=(1.05, 1), loc="upper left", borderaxespad
    =0.0)
    plt.title("Ad hoc dataset")
    plt.show()
```

Llamando a *plot_dataset* se podrá obtener una representación del espacio de características como el que se puede observar en la figura 20.

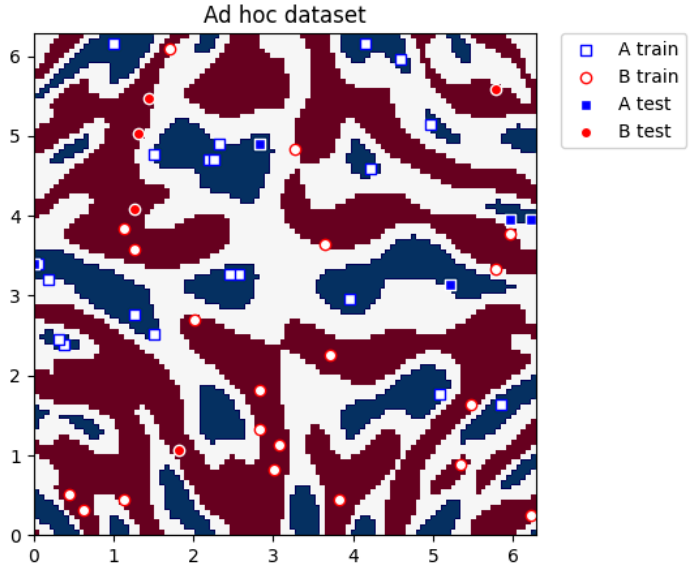


Figura 20: Espacio de características dado junto a datos de entrenamiento y de testeo [16]

A continuación, se computará el kernel. Para ello, se aplicarán las siguientes líneas:

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

adhoc_feature_map = ZZFeatureMap(feature_dimension=adhoc_dimension, reps=2,
    entanglement="linear")

adhoc_kernel = FidelityQuantumKernel(feature_map=adhoc_feature_map)
```

Donde *adhoc_feature_map* es el espacio de características definido en la figura 17 y *adhoc_kernel* será el que se puede observar en la figura 18.

Finalmente, para aplicar este kernel a una máquina de soporte vectorial se utilizará la función *QSVC*, la cual llama a una máquina de soporte vectorial clásica que utilizará el kernel cuántico:

```
from qiskit_machine_learning.algorithms import QSVC

qsvc = QSVC(quantum_kernel=adhoc_kernel)

qsvc.fit(train_features, train_labels)

qsvc_score = qsvc.score(test_features, test_labels)

print(f"QSVC classification test score: {qsvc_score}")
```

El resultado obtenido tiene una precisión del 100 %. Cabe destacar que estos datos están generados a partir de las funciones clásicas descritas en el kernel, y por tanto no es de extrañar su perfecta clasificación.