Universidad de Valladolid

Escuela de Ingeniería Informática

Estudio de QSVM, algoritmo de Machine Learning Cuántico

Proyecto conjunto con:

Universidad de Valladolid

Autor:

Guillermo Alonso Alonso de Linaje

Julio 2021

Índice general

A mi familia por su continuo apoyo para no rendirme nunca y luchar por mis metas, sin ellos no habría sido tan fácil llegar hasta donde estoy ahora. A mis tutores Juan José Álvarez y Juan Carlos García por su ayuda y perseverancia para sacar el proyecto adelante. Agradecer también a Javier Machín y Ginés Carrascal representando a IBM su gran apoyo ofrecido tanto dentro como fuera del trabajo para todo lo que necesitara.

Capítulo 1

Introducción

Uno de los problemas más comunes, y a la vez más complejos, a los que se ha enfrentado el ser humano son los problemas de clasificación. Distinguir ciertas especies vegetales a través de sus características, determinar si se concede o no un préstamos a un cliente o simplemente predecir si mañana lloverá, son ejemplos de este tipo de problemas. Con la llegada del aprendizaje automático junto con la cantidad de datos que somos capaces de recoger a día de hoy, se ha visto potenciada la capacidad de resolución. Entre los algoritmos más relevantes dentro de esta nueva rama de la informática podemos destacar los árboles de decisión, la clasificación lineal, las redes neuronales, técnicas de clustering o las máquinas de vector soporte (SVM). Dichos algoritmos se pueden dividir en dos clases: aprendizaje supervisado, en el que dispone de un conjunto de datos de entrada etiquetados (es decir, sabemos a qué clase final van a pertenecer); y no supervisados, en los que no tenemos información previa de cómo clasificar nuestros datos. A lo largo de este trabajo desarrollaremos un estudio de las máquinas de vector soporte, algoritmo de aprendizaje supervisado, y buscaremos potenciar su funcionamiento a través de un paradigma emergente de la que cada vez se oye hablar más, la computación cuántica.

Capítulo 2

Computación Cuántica

La computación cuántica es un paradigma de computación distinto al de la informática clásica (no cuántica) a la que estamos acostumbrados. El objetivo principal de este nuevo modo de cómputo es abarcar ciertos problemas que aparentemente un ordenador clásico no puede resolver o, al menos, no puede hacerlo en un tiempo razonable. Sin embargo, esta no es la única motivación para el desarrollo de la computación cuántica: estamos alcanzando el límite de escala en la integración de los chips clásicos. En escalas tan pequeñas (del orden de nanómetros) dichos chips dejan de funcionar adecuadamente ya que empiezan a aparecer propiedades cuánticas que afectan a su comportamiento.

La principal diferencia que encontramos entre ambos modos de computación es la unidad básica de información utilizada. Por un lado, los ordenadores clásicos trabajan con bits que pueden tomar el valor 0 o el valor 1, pero solo uno de estos dos estados a la vez. Por el otro lado, la unidad de información de un computador cuántico es el qubit que puede tener ambos estados además de un conjunto de estados intermedios (típicamente infinitos).

La referencia más popular para aprender sobre este campo es el libro de *Nielsen y Chuang* [10], no obstante, desarrollaremos algunos conceptos importantes que utilizaremos a lo largo del proyecto.

2.1. Qubits

Tal como comentábamos anteriormente, dos de los estados que puede tomar un qubit son el 0 y el 1, a los que llamaremos estados fundamentales y representaremos de la siguiente manera:

$$|0\rangle := \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad \text{y} \quad |1\rangle := \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Sin embargo, un qubit también puede tener cierta parte de $|0\rangle$ y otra de $|1\rangle$. Por lo tanto, un

estado arbitrario $|\phi\rangle$ se podría denotar de la forma

$$|\phi\rangle := \alpha\,|0\rangle + \beta\,|1\rangle = \begin{pmatrix}\alpha\\ \beta\end{pmatrix} \qquad \alpha, \beta \in \mathbb{C}$$

siendo ϕ (función de onda) solución de la ecuación de Schrödinger, de ahí que $\alpha, \beta \in \mathbb{C}$. La segunda propiedad de los qubits esta relacionada con la medición de los mismos. Un qubit puede estar en un estado arbitrario $|\phi\rangle$, sin embargo, a la hora de medir dicho qubit, si queremos observar cuál es su valor, éste colapsará hacia uno de los dos estados fundamentales (Colapso de la función de onda). La probabilidad con la que un estado colapsa hacia el estado 0 o 1 se puede calcular a partir de los valores α y β:

$$P(|0\rangle) = |\alpha|^2 \qquad P(|1\rangle) = |\beta|^2.$$

De dichas igualdades deducimos que todo estado debe tener norma unidad puesto que la probabilidad total debe ser igual a la suma de las probabilidades:

$$|\alpha|^2 + |\beta|^2 = 1.$$

Toda esta idea es fácilmente generalizable para n qubits donde los estados fundamentales corresponderían a los 2^n números binarios que podemos formar y cualquier estado en superposición se puede representar como

$$|\phi\rangle := \sum_{i=0}^{2^n-1} \alpha_i\,|i\rangle \qquad \alpha_i \in \mathbb{C} \tag{2.1}$$

donde $|i\rangle$ representa el estado fundamental cuyo valor puede representarse tanto en decimal como en binario natural. Por ejemplo $|5\rangle = |101\rangle$ por ser esa su descomposición en binario. Los coeficientes α_i podemos representarlos como coordenadas de un vector en la base $|i\rangle$ dado que el espacio de soluciones de la ecuación de Schrödinger es vectorial:

$$|\phi\rangle := \begin{pmatrix}\alpha_0\\ \alpha_1\\ ...\\ \alpha_{2^n-1}\end{pmatrix} \qquad \alpha_i \in \mathbb{C} \tag{2.2}$$

Del mismo modo que antes, podemos calcular la probabilidad de que un estado arbitrario colapse hacia $|i\rangle$:

$$P(|i\rangle) = |\alpha_i|^2 \qquad \forall i \in \{0, ..., 2^n - 1\}.$$

Además, al igual que veíamos con un qubit, la probabilidad total debe ser igual a la suma de las probabilidades, por lo que obtenemos:

$$\sum_{i=0}^{2^n-1} |\alpha_i|^2 = 1. \tag{2.3}$$

También se pueden definir estados de varios qubits como producto tensorial[1] de estados de un único qubit. Veamos el siguiente ejemplo:

$$\begin{gathered}(\alpha_0 |0\rangle + \alpha_1 |1\rangle) \otimes (\beta_0 |0\rangle + \beta_1 |1\rangle) = \\ = \alpha_0\beta_0 |0\rangle \otimes |0\rangle + \alpha_0\beta_1 |0\rangle \otimes |1\rangle + \alpha_1\beta_0 |1\rangle \otimes |0\rangle + \alpha_1\beta_1 |1\rangle \otimes |1\rangle = \\ = \alpha_0\beta_0 |00\rangle + \alpha_0\beta_1 |01\rangle + \alpha_1\beta_0 |10\rangle + \alpha_1\beta_1 |11\rangle\end{gathered}$$

Existen estados que no pueden descomponerse según esta regla como producto tensorial de estados simples. A dichos estados se les denomina *estados entrelazados* y juegan un papel muy importante en diferentes campos de la computación cuántica como la criptografía.

A lo largo de esta sección hemos definicido $|\phi\rangle$ como un vector columna (ket) , siendo esta la notación de Dirac. Siguiendo con esta notación podemos definir $\langle\phi|$ (bra) como el transpuesto del conjugado complejo de $|\phi\rangle$ tal y como se muestra a continuación:

$$|\phi\rangle := \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ ... \\ \alpha_{2^n-1} \end{pmatrix} \qquad \langle\phi| := \begin{pmatrix} \alpha_0^* & \alpha_1^* & ... & \alpha_{2^n-1}^* \end{pmatrix}.$$

Visto de esta forma, podemos interpretar el producto interno de dos estados ϕ y θ como $\langle\phi|\theta\rangle$ y en particular, en base a (2.3) obtenemos la igualdad:

$$\langle\phi|\phi\rangle = \sum_{i}^{2^n-1} \alpha_i\alpha_i^* = \sum_{i}^{2^n-1} |\alpha_i|^2 = 1.$$

Una vez definida la idea básica detrás de los qubits, pasemos a ver de qué manera podemos trabajar con ellos.

[1] El producto tensorial de dos vectores se calcula a partir de multiplicar cada uno de los elementos del primer vector por el segundo. Ejemplo: (2 1) $\otimes$ (0 -1) = (2(0 -1) 1(0 -1) = (2 -2 0 -1) consiguiendo un vector de longitud 4.

2.2. Puertas cuánticas

Volviendo a los ordenadores clásicos, por un lado tenemos bits, unidades básicas de información; y por otro, contamos con un conjunto de puertas lógicas que nos permiten llevar a cabo todas las operaciones necesarias. En los computadores cuánticos también disponemos de un conjunto de operadores que nos posibilitan interactuar con los qubits, a los que denominadermos puertas cuánticas, el cual tomará un estado de entrada y devolverá uno nuevo de salida.

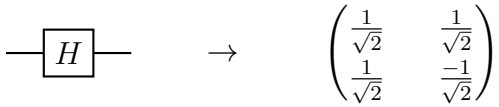
Recordemos que los estados cuánticos los podíamos representar como un vector columna de longitud 2^n siendo n el número de qubits. Teniendo esto en cuenta, al entender una puerta cuántica como un operador que modifica un estado, tiene sentido representar dichas puertas como matrices cuadradas de tamaño $2^n \times 2^n$:

$$|\text{output}\rangle = M_{2^n \times 2^n} |\text{input}\rangle .$$

Como podemos ver, el estado de salida correspondería al producto de la matriz M asociada a una puerta, por nuestro estado de entrada. Además dichas matrices tienen que ser unitarias, ya que debe cumplir que transformen vectores de norma unidad en vectores de norma unidad (ambos estados cuánticos). Las matrices unitarias, y por tanto todas las puertas cuánticas, tienen matriz inversa (que coincide con el conjugado complejo de la matriz), de donde deducimos que conociendo el estado de salida podemos calcular cuál fue el estado inicial. Este hecho, que no ocurre en la computación clásica [2], es lo que da lugar a la computación reversible. Veamos algunas de las puertas más utilizadas:

2.2.1. Puerta Hadamard

Esta puerta actúa sobre un único qubit y mapea $|0\rangle$ en $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ y $|1\rangle$ en $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$. Su representación es la siguiente:



$$\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{pmatrix}$$

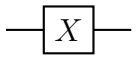
2.2.2. Puertas de Pauli

Las puertas de Pauli son un conjunto de puertas que al igual que la Hadamard actúan sobre un único qubit. Dichas puertas representan rotaciones de π radianes sobre la esfera de Bloch[3]. De

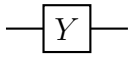
[2] No ocurre en la computación clásica con la que estamos familiarizados, no obstante; existen computadores reversibles que se acabaron descartando por su ineficiencia energética.

[3] Representación geométrica del espacio de estados puros de un sistema cuántico de dos niveles. El punto superior corresponde al $|0\rangle$ y el inferior al $|1\rangle$.

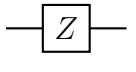
esta manera, la puerta X producirá la rotación respecto al eje X, y de igual forma, las puertas Y y Z, respecto a sus ejes.



$$\rightarrow \quad \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$



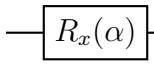
$$\rightarrow \quad \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$



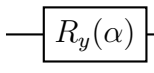
$$\rightarrow \quad \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

2.2.3. Puertas de rotación α

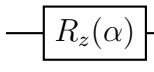
Las puertas de Pauli representan, como acabamos de mostrar, una rotación de π radianes respecto a cada uno de los ejes. Sin embargo, en diferentes ocasiones, necesitamos elegir el ángulo que deseamos rotar. Esto da lugar a las puertas de rotación:



$$\rightarrow \quad \begin{pmatrix} \cos\frac{\alpha}{2} & -i\sin\frac{\alpha}{2} \\ -i\sin\frac{\alpha}{2} & \cos\frac{\alpha}{2} \end{pmatrix}$$



$$\rightarrow \quad \begin{pmatrix} \cos\frac{\alpha}{2} & -\sin\frac{\alpha}{2} \\ \sin\frac{\alpha}{2} & \cos\frac{\alpha}{2} \end{pmatrix}$$

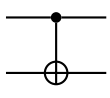


$$\rightarrow \quad \begin{pmatrix} e^{-i\frac{\alpha}{2}} & 0 \\ 0 & e^{i\frac{\alpha}{2}} \end{pmatrix}$$

siendo α los radianes que deseamos rotar el estado.

2.2.4. Puertas controladas

Las puertas controladas actúan sobre dos o más qubits, de los cuales uno o más controlan la operación. Distinguimos por tanto entre qubits de control y qubits objetivo. En el caso en el que todos los qubits de control tomen el valor $|1\rangle$, se aplicará la correspondiente puerta situada en los qubits objetivo. Por poner un ejemplo, la puerta C-NOT, representada de la siguiente manera:



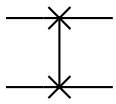
$$\rightarrow \quad \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

realiza una operación NOT sobre el qubit objetivo si el qubit de control es un 1. Por tanto, si

el qubit superior toma el valor $|1\rangle$ y el inferior $|0\rangle$, tras realizar esta operación, el primer qubit seguirá manteniendo su valor y al segundo qubit se le aplicará una puerta X (NOT) dando como salida $|1\rangle$.

2.2.5. SWAP

En la computación cuántica es muy frecuente el swap o cambio de valor entre qubits. Es decir, una operación que dado el estado de dos qubits distintos, los intercambie entre si.

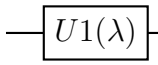


$$\rightarrow \quad \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

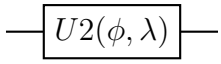
Si, por ejemplo, aplicamos este operador al estado $\frac{|0\rangle+|1\rangle}{\sqrt{2}} \otimes |0\rangle$, obtendremos como resultado $|0\rangle \otimes \frac{|0\rangle+|1\rangle}{\sqrt{2}}$.

2.2.6. Puertas universales U1, U2 y U3

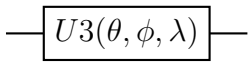
$U1$, $U2$ y $U3$ son tres puertas, cada una es generalización de la anterior, con la que se puede definir cualquier otra puerta que actúe sobre un único qubit.



$$\rightarrow \quad \begin{pmatrix} 1 & 0 \\ 0 & e^{i\lambda} \end{pmatrix}$$



$$\rightarrow \quad \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -e^{i\lambda} \\ e^{i\phi} & e^{i(\lambda+\phi)} \end{pmatrix}$$



$$\rightarrow \quad \begin{pmatrix} \cos\frac{\theta}{2} & -e^{i\lambda}\sin\frac{\theta}{2} \\ e^{i\phi}\sin\frac{\theta}{2} & e^{i(\lambda+\phi)}\cos\frac{\theta}{2} \end{pmatrix}$$

La puerta CNOT junto a U3 forman un conjunto universal, es decir, se puede formar cualquier otra puerta a partir de la combinación de éstas. La construcción de U1 y U2 son casos particulares que aparecen de forma natural a partir de U3.

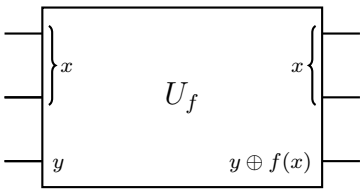
Por poner un ejemplo de esto, se puede comprobar que $U3(0, 0, \pi) = Z$.

2.3. Algoritmo de Deutsch-Jozsa

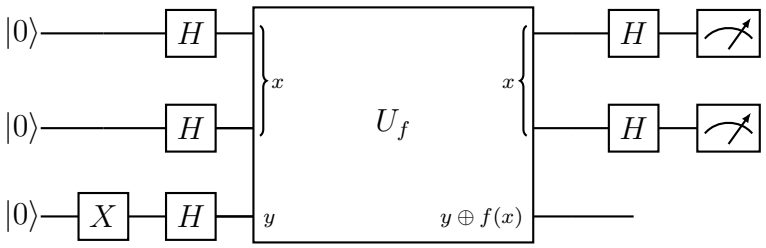
A través del siguiente ejemplo veremos una prueba de la superioridad de la computación cuántica. El algoritmo de Deusch-Jozsa [2] busca determinar si dada una función $f : \{0, ..., 2^n - 1\} \rightarrow \{0, 1\}$

es constante (es decir, $f(x) = 0$ o $f(x) = 1\ \forall x \in \{0, .., 2^n - 1\}$) o es balanceada (la mitad de los puntos van a 0 y la otra mitad a 1). En un supuesto clásico en el que pudieramos ir preguntando a f por la imagen de diferentes puntos x, en el peor de los casos necesitaríamos hacer $2^{n-1} + 1$ preguntas (la mitad más 1), ya que podría ser que preguntáramos justo por todos los elementos de un mismo conjunto. Sin embargo, y aquí radíca la fuerza del algoritmo de Deutsch-Jozsa, en la computación cuántica sería necesaría realizar una única llamada a la función.

Por aclarar la notación, denotaremos x al valor entre 0 y 2^n-1 que queremos evaluar, y escribiremos como $|x\rangle$ al estado formado por su representación binaria. Por poner un ejemplo de esto, si $x = 6$ entonces $|x\rangle$ o $|6\rangle$ haría referencia a $|110\rangle$, que es equivalente a $|1\rangle \otimes |1\rangle \otimes |0\rangle$. En primer lugar, recordar que toda puerta tiene que ser reversible (2.2), y sin embargo esa función f no es biyectiva. Para evitar este problema, este tipo de funciones se construyen a través de un operador U_f que transforma $|x\rangle |y\rangle$ a $|x\rangle |y \oplus f(x)\rangle$[4]. De este modo, tomando $y = 0$ y observando el último qubit, obtendríamos $|f(x)\rangle$:



Poniendo un ejemplo, si $x = 3 = (11)_2$, tomando $y = 0$, tendríamos que la entrada es $|110\rangle$ y la salida $|11f(3)\rangle$. El objetivo será determinar que clase de función es, utilizando una única vez el operador U_f. Para ello, el algoritmo establece que debemos inicializar x a 0, incluir puertas Hadamard en la entrada y salida de los qubits con los que representamos x, y tomar $y = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$. Quedaría el circuito como:



donde estamos definiendo x con los primeros n qubits haciendo un total de $n + 1$ qubits para el circuito. Definido de esta forma, si observamos que todos los qubits son $|0\rangle$ ($|x = 0\rangle = |0\rangle^n$ querría decir que f es constante y si no, f sería balanceada. Veamos por qué se produce este fenómeno. Inicialmente el estado es $|0\rangle^{n+1}$ que tras aplicar la primera puerta X se convertirá en $|0\rangle^n |1\rangle$. Aplicando la primera columna de puertas Hadamard, el estado se convertirá en la siguiente expresión[5]:

[4]El operador $\oplus$ corresponde a la suma binaria.

[5]Viendo un ejemplo para $n = 2$, $H^2 |00\rangle = H |0\rangle H |0\rangle = \left(\frac{|0\rangle+|1\rangle}{\sqrt{2}}\right)\left(\frac{|0\rangle+|1\rangle}{\sqrt{2}}\right) = \frac{1}{2}(|00\rangle + |01\rangle + |10\rangle + |11\rangle) = \frac{1}{2}(|0\rangle + |1\rangle + |2\rangle + |3\rangle)$ tal y como queríamos comprobar

$$H^n |0\rangle^n H |1\rangle = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}} \tag{2.4}$$

El siguiente paso es introducir el estado por la puerta U_f, que por linealidad de U_f llegamos a que:

$$U_f \left(\frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} |i\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) = \frac{1}{\sqrt{2^{n+1}}} \sum_{i=0}^{2^n-1} (U_f |i\rangle |0\rangle - U_f |i\rangle |1\rangle)$$

Ahora por la definición dada del operador U_f sabemos que $U_f |x\rangle |y\rangle = |x\rangle |1 \oplus f(x)\rangle$ por lo que:

$$\frac{1}{\sqrt{2^{n+1}}} \sum_{i=0}^{2^n-1} (U_f |i\rangle |0\rangle - U_f |i\rangle |1\rangle) = \frac{1}{\sqrt{2^{n+1}}} \sum_{i=0}^{2^n-1} (|i\rangle |f(i)\rangle - |i\rangle |1 \oplus f(i)\rangle).$$

A partir de este punto tenemos dos opciones, que $f(i) = 0$ y en este caso podríamos agrupar los términos como $|i\rangle (|0\rangle - |1\rangle)$ o que $f(i) = 1$, lo que formaría el estado $|i\rangle (-|0\rangle + |1\rangle)$. Por tanto, independiente del valor de $f(i)$, podríamos agrupar ambas situaciones como $|i\rangle (-1)^{f(i)}(|0\rangle - |1\rangle)$, llegando a la igualdad siguiente:

$$\frac{1}{\sqrt{2^{n+1}}} \sum_{i=0}^{2^n-1} (|i\rangle |f(i)\rangle - |i\rangle |1 \oplus f(i)\rangle) = \frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} (-1)^{f(i)} |i\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

El estado anterior corresponde al momento en el que acabamos de pasar por la puerta U_f, finalmente debemos volver a pasar por puertas Hadamard como se indica en el esquema del circuito (2.3). Para continuar usaremos la siguiente identidad de gran utilidad:

$$H^n |x\rangle = \frac{1}{\sqrt{2^n}} \sum_{z \in \{0,1\}^n} (-1)^{xz} |z\rangle , \tag{2.5}$$

donde xz es una simplificación de $x_0 z_0 \oplus x_1 z_1 \oplus ...$[6] y que aplicándolo a (2.3), conseguimos la expresión:

$$H^n \left(\frac{1}{\sqrt{2^n}} \sum_{i=0}^{2^n-1} (-1)^{f(i)} |i\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} (-1)^{f(x)} \frac{1}{\sqrt{2^n}} \sum_{z \in \{0,1\}^n} (-1)^{xz} |z\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) =$$

$$= \sum_{x,z \in \{0,1\}^n} \frac{(-1)^{f(x)+xz} |z\rangle}{2^n} \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right) \tag{2.6}$$

[6] x_i no es más que el i-ésimo digito de la representación binaria de x, por lo que tomará únicamente los valores 0 o 1.

Dado este estado final ya podemos calcular la probabilidad de observar todo ceros a través de la amplitud de este estado, es decir:

$$\sum_{x\in\{0,1\}^n} \frac{(-1)^{f(x)}}{2^n} \tag{2.7}$$

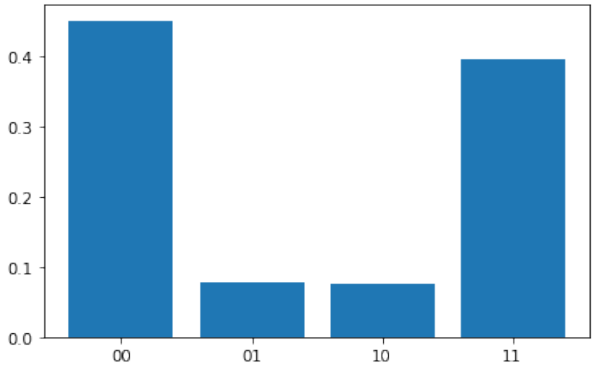
A la vista de esto, si f fuese constante, la amplitud de dicho estado sería 1, por lo que siempre observaríamos $|0\rangle^n$. Sin embargo, si f fuese balanceada, la amplitud sería 0 y nunca observaremos $|0\rangle^n$. Obsérvese que con una sola medida (invocación a $f(x)$) sabemos la solución en contraposición con las $2^{n-1}+1$ invocaciones a $f(x)$ que se necesitarían en una situación clásica para el caso más desfavorable.

2.4. Implementación

En esta sección mostraremos cómo podemos implementar un circuito en un computador cuántico real. En concreto se ha utilizado Pennylane [1], una librería creada específicamente para la resolución de problemas del campo del Quantum Machine Learning. La construcción y ejecución del circuito es muy sencilla, veámoslo a través del siguiente fragmento de código:

```
1  import pennylane as qml
2  import matplotlib.pyplot as plt
3
4  dev = qml.device("qiskit.ibmq", wires = 2, shots = 1000, backend = "ibmq_16_melbourne")
5
6  @qml.qnode(dev)
7  def circuit():
8      qml.Hadamard(wires = 0)
9      qml.CNOT(wires = [0,1])
10     return qml.probs(range(2))
11
12 p = circuit()
13
14 plt.bar(["00","01","10","11"],p)
15 plt.show()
```

Como podemos comprobar lo primero que hacemos es importar la librería en cuestión y una auxiliar para posteriormente representar los datos. En la línea cuatro estamos estableciendo el servicio elegido para ejecutarlo, donde podemos ver que en este caso llamaremos al ordenador 'Melbourne' de IBM. El parámetro 'wires' hace referencia al número de qubits que necesitaremos y 'shots' a cuantas veces realizaremos las mediciones. Después de esto en la línea 7 definimos el circuito añadiendo diferentes puertas cuánticas y pidiendo que lo que nos devuelva sea la probabilidad de observar cada estado. El decorador '@qml.qnode(dev)' se encarga de asociar al circuito el servicio que hemos elegido ya que de no ser así no se sabría donde enviarlo. Por tanto, al ejecutar la línea 12 y llamar al circuito se llevará a cabo un proceso de compilación de las instrucciones que hemos definido y se enviarán al ordenador de IBM en un formato legible para él. Tras ejecutarse se nos devolverán los resultados y los mostraremos por pantalla obteniendo un histograma de frecuencias como el siguiente:



Desde un punto de vista teórico tendríamos que haber obtenido que el estado $|00\rangle$ y el $|11\rangle$ tienen exactamente un 50 % de probabilidades ,sin embargo, se aprecia que con cierta probabilidad vamos a ver estados que no corresponden con el circuito. A día de hoy los ordenadores cuánticos están continuamente siendo calibrados para intentar reducir este tipo de errores experimentales. También se aplican diferentes técnicas, a nivel software, para reducir estos errores. Es frecuente desde un punto de vista didáctico utilizar simuladores cuánticos, es decir, programas hechos en ordenadores tradicionales que modelan el comportamiento de estos circuitos. Aunque no escalan bien, son muy utilizados para realizar pequeñas pruebas sin este tipo de errores para ilustrar comportamientos cuánticos de manera simulada.

Capítulo 3

Máquinas de Vector Soporte

El aprendizaje automático es un campo dentro de la ciencia de la computación cuyo principal objetivo es desarrollar técnicas para que las computadoras aprendan. Diremos que un computador aprende cuando su rendimiento mejora con la experiencia, es decir, adquiere habilidades con las que no contaba en su estado inicial. Dentro del aprendizaje automático encontramos el aprendizaje por refuerzo, técnica para deducir una función a partir de unos datos de entrenamiento. Además dichos datos de entrenamiento cuentan con una estructura diferenciada: por un lado los datos de entrada, y por otro; los resultados deseados.

El algoritmo con el que vamos a trabajar se denomina SVM (Máquina de Vector Soporte) [3]. Lo podemos situar dentro del aprendizaje supervisado. Este algoritmo está relacionado con problemas de clasificación, es decir; dado un conjunto de muestras, podemos etiquetar las clases y entrenar nuestra máquina para que prediga la clase de una nueva muestra.

3.1. Idea intuitiva

Las Máquinas de Vector Soporte son un clasificador lineal, ya que pretenden separar las clases a través de hiperplanos (generalización de las rectas en varias dimensiones). Por simplificar la idea, supondremos que tenemos únicamente dos clases y que además son separables, es decir; existe un hiperplano capaz de dejar a un lado todos los elementos de una clase y al otro lado el resto. Como podemos ver en la imagen, existen infinidad de posibles planos que nos servirían para separar nuestros conjuntos, sin embargo, para las SVMs no es suficiente con ecntrar un plano que cumpla esta condición: intentan encontrar aquel que maximice las distancias entre las dos clases.

La distancia entre nuestro hiperplano y los puntos más cercanos de ambas clases recibe el nombre de *margen*, por lo que también denominamos las SVM como clasificadores de *margen* máximo. Por otro lado, dichos puntos que se encuentran más próximos al hiperplano, se conocen como *puntos soporte* y son los que realmente determinan la posición final de éste. Teniendo esta idea

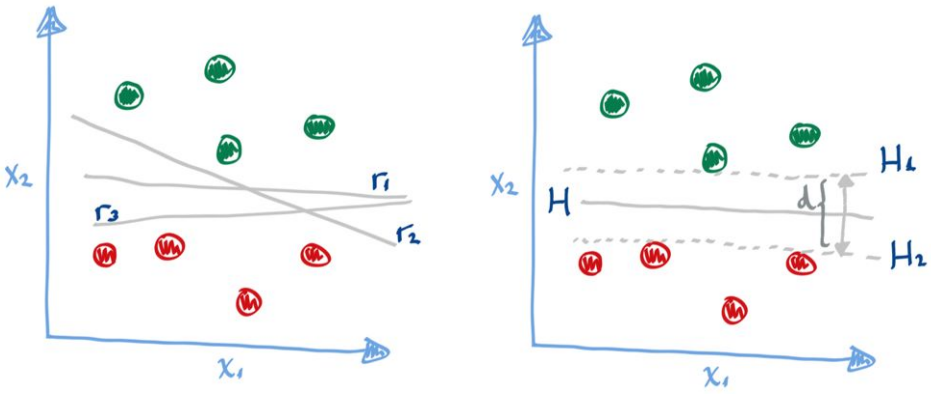


Figura 3.1: En la ilustración de la izquierda se puede observar 3 posibles hiperplanos que separan completamente las dos clases. Al lado derecho se muestra el hiperplano que maximiza la distancia entre ambas clases.

en mente, el algoritmo que hay detrás de las SVM proporciona un método para encontrar estos puntos críticos y a partir de ellos, obtener el hiperplano solución.

3.2. Formulación

En esta sección detallaremos el proceso matemático que se sigue para encontrar los vectores soporte. Sea $H = \{\vec{x} \in \mathbb{R}^n / \vec{x} \cdot \vec{w} + b = 0\}$ el hiperplano buscado definido por $\vec{w} \in \mathbb{R}^n$ y $b \in \mathbb{R}$. Definamos por otro lado Ω como el conjunto de nuestros datos o vectores de características y $f(\vec{x})$ como el valor de la clase asociado a cada $\vec{x} \in \Omega$ que, al trabajar únicamente con dos clases, diremos que $f(\vec{x}) \in \{1, -1\}$. De esta forma sería sufiente econtrar unos parámetros $\vec{w}$ y b que cumplan [1]:

$$f(\vec{x})(\vec{w} \cdot \vec{x} + b) \geq 0. \tag{3.1}$$

Ahora bien, lo que vamos a querer maximizar es la longitud del *margen*, y para ello vamos a definir los planos que lo limitan:

$$H_1 = \{\vec{z} \in \mathbb{R}^n / \vec{z} \cdot \vec{w} + b = 1\}$$
$$H_2 = \{\vec{z} \in \mathbb{R}^n / \vec{z} \cdot \vec{w} + b = -1\}$$

y el plano H que divide el *margen* en dos partes iguales:

[1] Recordemos que dada la ecuación de un plano de la forma $\vec{w}.\vec{z} + b = 0$ tendremos que $\vec{w}.\vec{x} + b > 0$ si $\vec{x}$ esta a un lado del plano y $\vec{w}.\vec{x} + b < 0$ si está al otro lado. Lo que da lugar a (3.1)

$$H = \{\vec{z} \in \mathbb{R}^n / \vec{z} \cdot \vec{w} + b = 0\}$$

Por hacernos una idea, H_1 y H_2 corresponderían con los planos marcados en la imagen derecha de la figura 1 en línea discontinua. Teniendo esto definido, podemos calcular la distancia entre los planos de la siguiente manera:

Sea $\vec{y} \in H_2$ y sea

$$\vec{z} := \vec{y} + \frac{d}{2}\frac{\vec{w}}{\|\vec{w}\|} \in H.$$

donde $\frac{d}{2}$ es la distancia entre el plano H_2 y H. Se puede definir $\vec{z}$ de esta manera ya que los planos son paralelos y $\vec{w}$ es un vector ortogonal al plano. Por tanto, si multiplicamos a ambos lados por $\vec{w}$ y dividimos por la norma, obtenemos que:

$$\frac{\vec{z} \cdot \vec{w}}{\|\vec{w}\|} = \frac{\vec{y} \cdot \vec{w}}{\|\vec{w}\|} + \frac{d}{2}$$

y sabiendo que $\vec{y} \in H_2$ y $\vec{z} \in H$ llegamos a:

$$\frac{-b}{\|\vec{w}\|} = \frac{-1-b}{\|\vec{w}\|} + \frac{d}{2}$$

obteniendo así que la distancia entre H_2 y H es $\frac{1}{\|\vec{w}\|}$. Por simetría, la distancia de H_1 a H será la misma, luego el problema buscará maximizar $\frac{2}{\|\vec{w}\|}$ o lo que es lo mismo [2]:

$$\text{minimizar } \frac{\|\vec{w}\|^2}{2}$$

$$\text{sujeto a } \quad f(\vec{x_i})(\vec{w} \cdot \vec{x_i} + b) \geq 1 \qquad \forall \vec{x_i} \in \Omega,$$

ya que buscando maximizar la distancia entre planos, garantizaríamos la correcta división de la muestra y que no haya ningún punto dentro del margen establecido. Este es un problema cuadrático convexo que podemos resolver haciendo uso de la versión extendida de los multiplicadores de Lagrange, KKT(Karush-Kuhn-Tucker). Una de las grandes ventajas de usar máquinas soporte rádica en que, en un problema cuadrático convexo, existe un único mínimo local que coincide con

[2]Esto se debe a que hacer grande un número positivo x es equivalente a hacer pequeño su inverso x^{-1}. Además al ser positivo dará lo mismo minimizar un número que su cuadrado ya que si $x < y$ entonces $x^2 < y^2$.

el global, por lo que no nos encontraremos con el problema de caer en mínimos locales relativos, principal problema de las redes neuronales. En base a la función objetivo y las restricciones definidas, el problema es equivalente a encontrar un mínimo en la siguiente función:

$$L = \frac{\vec{w} \cdot \vec{w}}{2} + \sum_{i=0}^{m} \alpha_i [f(\vec{x_i})(\vec{w} \cdot \vec{x_i} + b) - 1], \tag{3.2}$$

siendo en este caso α_i los coeficientes de Lagrange. Si lo que estamos buscando es un mínimo de dicha función, ha de cumplirse que la derivada con respecto a $\vec{w}$ y b sea nula, es decir:

$$\frac{\partial L}{\partial \vec{w}} = \vec{w} + \sum_{i=0}^{m} \alpha_i f(\vec{x_i}) \vec{x_i} = 0 \tag{3.3}$$

$$\frac{\partial L}{\partial b} = \sum_{i=0}^{m} \alpha_i f(\vec{x_i}) = 0 \tag{3.4}$$

y sustituyendo estas igualdades en la ecuación (3.2) conseguimos llegar al problema dual[3],

$$L' = -\sum_{i=0}^{m} \alpha_i + \frac{1}{2} \sum_{i=0}^{m} \sum_{j=0}^{m} \alpha_i \alpha_j f(\vec{x_i}) f(\vec{x_j}) \vec{x_i} \cdot \vec{x_j} \tag{3.5}$$

Notemos que al ser las restricciones desigualdades, siguiendo la formulación KKT, en este caso, se añade que se debe cumplir la siguiente condición:

$$\alpha_i [f(\vec{x_i})(\vec{w}.\vec{x_i} + b) - 1] = 0. \tag{3.6}$$

De aquí deducimos que para aquellos puntos que no pertenezcan a los hiperplanos soporte, α_i será 0. Una vez que tenemos estos valores $\vec{\alpha}$, procederemos a recuperar los valores $\vec{w}$ y b. En primer lugar basándonos en (3.3) podemos calcular facilmente $\vec{w}$ como:

$$\vec{w} = -\sum_{i=0}^{m} \alpha_i f(\vec{x_i}) \vec{x_i}.$$

Para recuperar b tenemos que para una serie de puntos (aquellos que pertenecen a los hiperplanos soporte) se cumple que $f(\vec{x_i})(\vec{w}.\vec{x_i}+b)-1 = 0$. Multipllicando a ambos lados por $f(\vec{x_i})$ y despejando la incognita obtenemos que:

$$b = f(\vec{x_i}) - \vec{w}.\vec{x_i}.$$

[3]Llegamos a esta expresión dándonos cuenta de que $\sum_{i=0}^{m} \alpha_i f(\vec{x_i}) \vec{w}.\vec{x_i}$ es constante ya que es equivalente a $\vec{w}.\vec{w}$.

Podríamos tomar por tanto b como la media de dichos valores para cada uno de los vectores soporte:

$$b = \frac{1}{S}\sum_{i=1}^{S}(f(\vec{x_i}) - \vec{w}.\vec{x_i})$$

donde S es el número de vectores soporte y los $\vec{x_i}$ dichos vectores. La gran ventaja que existe con dicha formulación será tratada más adelante, para lo cuál daremos previamente una breve idea a cerca de los kernels.

3.3. Kernels

Para todo el procedimiento explicado hasta ahora hemos tenido en cuenta una suposición muy fuerte, los datos eran linealmente separables, es decir; hemos supuesto que existe un hiperplano que divide totalmente ambas clases. A través de los kernels podemos lidiar con situaciones en las que esto no se dé. La idea consiste en mapear los datos a un espacio de dimensión mayor en el que sí sean separables.

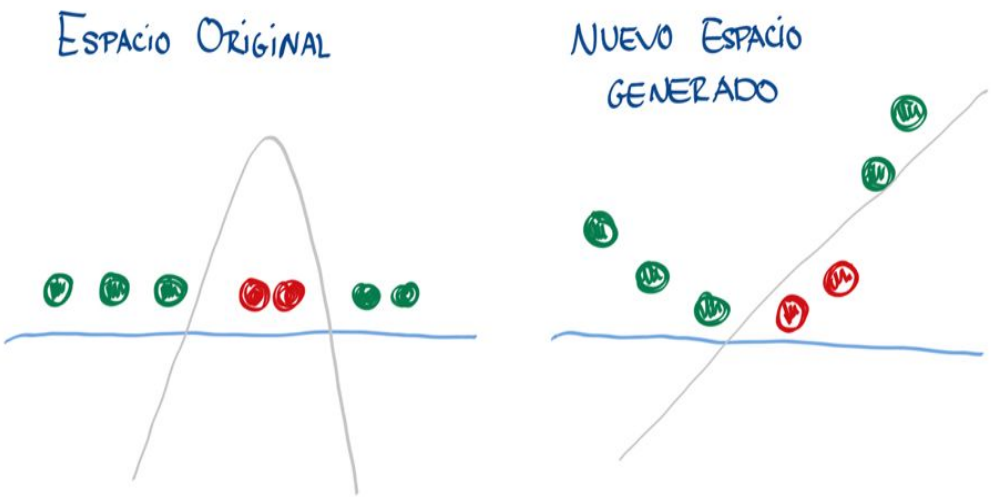


Figura 3.2: Se puede apreciar como inicialmente no existe una separación lineal pero tras aumentar la dimensión con un mapeo, podemos encontrar la división.

Ahora bien, definamos ϕ como la función kernel que mapea cada elemento en uno de dimensión mayor. Siguiendo el ejemplo de la ilustración, tendríamos $\phi : x \to (x, x^2)$ y sustituyendo en (3.5), obtenemos el mismo problema salvo con una pequeña modificación:

$$L = -\sum_{i=0}^{m} \alpha_i + \frac{1}{2} \sum_{i=0}^{m} \sum_{j=0}^{m} \alpha_i \alpha_j f(\vec{x_i}) f(\vec{x_j}) \phi(\vec{x_i}) \cdot \phi(\vec{x_j}) \tag{3.7}$$

y, por simplificación, podemos definir la función kernel como:

$$K(\vec{x_i}, \vec{x_j}) := \phi(\vec{x_i}) \cdot \phi(\vec{x_j}). \tag{3.8}$$

Nótese que visto de esta manera no nos hace falta calcular explícitamente los valores $\phi(\vec{x_i})$, sino que solo sería necesario obtener el producto interno de dos elementos transformados por el mapeo ϕ.

No obstante, nos encontramos con una complicación en este punto: a la hora de obtener el valor $\vec{w}$ tenemos que por (3.2):

$$\vec{w} = -\sum_{i=0}^{m} \alpha_i f(\vec{x_i}) \phi(\vec{x_i}), \tag{3.9}$$

y, por tanto, sí que deberíamos calcular explícitamente los valores de ϕ. Sin embargo, esto es algo que podemos evitar teniendo en cuenta que nuestro objetivo final es clasificar, no calcular el valor de $\vec{w}$. Es por ello que sería suficiente poder obtener $\vec{w}.\phi(\vec{x_j}) + b$ para un nuevo valor $\vec{x_j}$, y en este caso, si el resultado es mayor que 0, correspondería a la clase 1 o si fuese menor, a la clase -1 (por (3.1)).

Entonces, la idea fundamental está en darse cuenta de que:

$$\vec{w}.\phi(\vec{x_j}) = -\sum_{i=0}^{m} \alpha_i f(\vec{x_i}) \phi(\vec{x_i}).\phi(\vec{x_j}) = -\sum_{i=0}^{m} \alpha_i f(\vec{x_i}) K(\vec{x_i}, \vec{x_j}) \tag{3.10}$$

Es decir, aunque necesitábamos calcular $\phi(x_i)$ para estimar $\vec{w}$, no nos hace falta para conseguir el valor $\vec{w}.\phi(x_j)$ tal y como podemos ver en la expresión anterior. De forma análoga podemos obtener b y por tanto, determinar la clase a la que pertenece el elemento x_j sin calcular en ningún momento los valores $\phi(\vec{x})$.

El procedimiento es este punto suele ser precalcular todos los productos $K(\vec{x_i}, \vec{x_j})\ \forall i, j \in \{1, ..., n\}$. De esta forma construimos lo que se denomina la *matriz de Gram*, estando en la fila i-ésima y en la columna j-ésima el elemento $K(\vec{x_i}, \vec{x_j})$. En (3.3) podemos ver el código implementado para el cálculo de la matriz.

El procedimiento consiste en recorrer todos los elementos de un vector $X1$ y otro elemento $X2$ tal y como se aprecia en las líneas 80 y 81. Después calculamos el producto escalar entre ambos (línea 87) y lo insertamos en su posición correspondiente (línea 89). El resto del código no comentado se

```
73      def KernelGramMatrixFull(X1, X2):
74
75          simp = False
76          if X1 is X2:
77              simp = True
78
79          gram_matrix = np.zeros((X1.shape[0], X2.shape[0]))
80          for i, x1 in enumerate(X1):
81              for j, x2 in enumerate(X2):
82                  x1 = x1.flatten()
83                  x2 = x2.flatten()
84
85                  if simp and i < j:
86                      continue
87                  p = scalar_product(x1,x2,w)
88                  if simp:
89                      gram_matrix[j, i] = p
90
91                  gram_matrix[i, j] = p
92          return gram_matrix
93
```

Figura 3.3: Código para la creación de la matriz de Gram.

encarga de simplificar el proceso ya que para $X1 = X2$ tendríamos una matriz simétrica[4] y haría falta calcular la mitad de los valores.

[4]Esto se debe a que los productos escalares que definiremos cumplirán que $K(\vec{x_i}, \vec{x_j}) = K(\vec{x_j}, \vec{x_i})$

Capítulo 4

QSVM

El aprendizaje automático cuántico (Quantum Machine Learning) es una área de investigación emergente que podemos situar entre la mecánica cuántica y la informática. Busca aprovechar el paralelismo intrínseco de la computación cuántica para proporcionar soluciones al análisis de grandes volúmenes de datos, que necesitarán codificarse para ser accesibles por estos computadores.

A lo largo de este trabajo vamos a estudiar el funcionamiento de la versión cuántica del SVM (QSVM) la cual está recibiendo mucho atención en trabajos recientes [8] [11] [7]. Para ello, el camino seguido consistirá en obtener rendimiento a la hora de calcular el valor de la función Kernel definida en (3.8) teniendo en cuenta que entrenar un SVM clásico es útil cuando el producto interno de los vectores de características pueden ser calculados fácilmente. Visto de esta manera, obtendremos la ventaja cuántica cuando el Kernel buscado no se pueda estimar de forma clásica de forma eficiente.

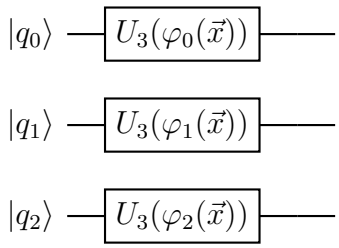
4.1. Mapa de características cuántico

Lo que nosotros propondremos será un clasificador basado en circuitos cuánticos para lo cual, el primer paso será mapear nuestros datos $\vec{x} \in \Omega$ en el circuito. Este es un proceso que se lleva a cabo a través del mapa de características que podemos definir con una puerta $U_{\phi(\vec{x})}$ siendo $\phi : \vec{x} \to \mathbb{R}^n$ una función kernel asociada. Dicha puerta $U_{\phi(\vec{x})}$ se aplica a un estado inicial, que por convenio se suele escoger $|0\rangle^{\otimes n}$, es decir; la representación cuántica del vector $\vec{x}$ en el circuito será $U_{\phi(\vec{x})} |0\rangle^{\otimes n}$. A continuación mostraremos diferentes mapas de características que nos podemos encontrar.

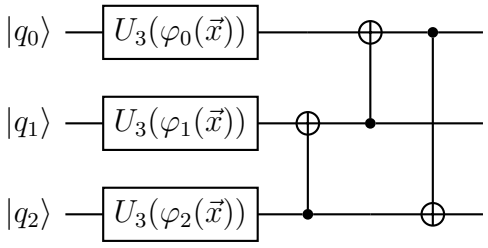
En primer lugar supondremos que los datos serán transformados a través de rotaciones en los qubits individuales. Para ello, los ángulos de rotación de cada qubit pueden ser elegidos a través de una función no lineal $\varphi : \vec{x} \to (0, 2\pi] \times (0, 2\pi] \times [0, \pi]$, por lo que el mapa de características quedaría definido de la siguiente manera:

$$\vec{x} \to |\phi_i(\vec{x})\rangle = U(\varphi_i(\vec{x}))\,|0\rangle \qquad \forall i \in 1, 2, ..., n. \tag{4.1}$$

Un ejemplo de mapa de características producto prodría ser este que tenemos aquí para el caso concreto de 3 qubits, donde vamos a utilizar la puerta genérica U_3:



No obstante, es frecuente encontrar mapas de características que juegen con el entrelazamiento con el objetivo de aprovechar el potencial de la computación cuántica. Una forma sencilla de añadir este entrelazamiento sería a través de una serie de puertas $CNOT$ tal y como se muestra a continuación:

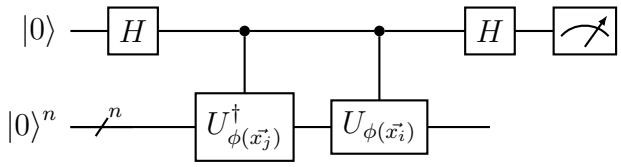


4.2. Cálculo del producto escalar

Como hemos visto anteriormente, dado un estado $\vec{x}$, a través de un mapa de características lo enviamos al nuevo estado $U_{\phi(\vec{x})}\,|0\rangle^{\otimes n}$. Pero tal cual está definida la ecuación del algoritmo, no nos hace falta el estado en sí, lo que estamos buscando es el producto escalar de la imagen de dos estados iniciales. De forma más precisa: dados dos estados $\vec{x_i}$ y $\vec{x_j}$, se busca el valor $\langle 0|^{\otimes n} U^{\dagger}_{\phi(\vec{x_j})} U_{\phi(\vec{x_i})}\,|0\rangle^{\otimes n}$ (siendo $U^{\dagger}$ la traspuesta del conjugado complejo de U). No obstante, este valor podría ser complejo y de cara al algoritmo solo nos interesa que tome valores reales, por lo que intentaremos obtener la parte real de dicho valor[1].

[1]Otra opción sería calcular el módulo del estado que obtengamos, sin embargo hemos decidido quedarnos con la parte real ya que consideramos importante que puedan salir número negativos en la expresión del valor esperado $\langle 0|^{\otimes n} U^{\dagger}_{\phi(\vec{x_j})} U_{\phi(\vec{x_i})}\,|0\rangle^{\otimes n}$.

Para calcular $\Re(\langle 0|^{\otimes n} U^\dagger_{\phi(\vec{x_j})} U_{\phi(\vec{x_i})} |0\rangle^{\otimes n})$ nos ayudaremos de lo que se conoce como el Hadamard Test, cuya estructura es la siguiente:



De esta forma, obtendríamos que $\Re(\langle 0|^{\otimes n} U^\dagger_{\phi(\vec{x_j})} U_{\phi(\vec{x_i})} |0\rangle^{\otimes n}) = P(0) - P(1)$ siendo $P(0)$ y $P(1)$ la probabilidad de obtener $|0\rangle$ y $|1\rangle$ respectivamente al medir sobre el primer qubit. Veamos la demostración de esta afirmación:

Inicialmente comenzamos por el estado $|0\rangle^{\otimes n+1}$, el cual, tras aplicar la puerta H sobre el primer qubit, se nos quedará como

$$|0\rangle^{\otimes n} \frac{|0\rangle + |1\rangle}{\sqrt{2}}$$

Por simplicidad llamaremos V a $U^\dagger_{\phi(\vec{x_j})} U_{\phi(\vec{x_i})}$, por tanto, tras realizar las dos puertas controladas, se nos quedará el siguiente estado:

$$\frac{(V|0\rangle^{\otimes n})|1\rangle}{\sqrt{2}} + \frac{(|0\rangle^{\otimes n})|0\rangle}{\sqrt{2}},$$

y volviendo a aplicar una puerta H sobre el primer qubit llegamos al estado final:

$$\frac{(V|0\rangle^{\otimes n})|0\rangle - (V|0\rangle^{\otimes n})|1\rangle}{2} + \frac{(|0\rangle^{\otimes n})|0\rangle + (|0\rangle^{\otimes n})|1\rangle}{2}.$$

Agrupando respecto al qubit de medición se nos quedaría de la siguiente manera:

$$\frac{(V|0\rangle^{\otimes n} + |0\rangle^{\otimes n})|0\rangle - (V|0\rangle^{\otimes n} - |0\rangle^{\otimes n})|1\rangle}{2}.$$

Por tanto, ya podemos calcular la probabilidad de obtener 0 y 1 respecto al primer qubit tomando el módulo de sus coeficientes, es decir:

$$P(0) = \| \frac{(V|0\rangle^{\otimes n} + |0\rangle^{\otimes n})}{2} \|^2 = \frac{(\langle 0|^{\otimes n} V^\dagger + \langle 0|^{\otimes n})(V|0\rangle^{\otimes n} + |0\rangle^{\otimes n})}{4} =$$

$$= \frac{2 + \langle 0|^{\otimes n} V^\dagger |0\rangle^{\otimes n} + \langle 0|^{\otimes n} V |0\rangle^{\otimes n}}{4},$$

dado que $V^\dagger V = I$ por ser V unitaria y ,por tanto, coincidir $V^\dagger$ con la inversa de V. De forma análoga podemos calcular la probabilidad de obtener 1 como:

$$P(1) = \|\frac{(V|0\rangle^{\otimes n} - |0\rangle^{\otimes n})}{2}\|^2 = \frac{(\langle 0|^{\otimes n} V^\dagger - \langle 0|^{\otimes n})(V|0\rangle^{\otimes n} - |0\rangle^{\otimes n})}{4} =$$

$$= \frac{2 - \langle 0|^{\otimes n} V^\dagger |0\rangle^{\otimes n} - \langle 0|^{\otimes n} V |0\rangle^{\otimes n}}{4},$$

y restando ambas probabilidad podemos ver que:

$$P(0) - P(1) = \frac{\langle 0|^{\otimes n} V^\dagger |0\rangle^{\otimes n} + \langle 0|^{\otimes n} V |0\rangle^{\otimes n}}{2}. \tag{4.2}$$

Para finalizar solo habría que darse cuenta de que la suma de un número imaginario y su conjugado nos da el doble de su parte real, por lo que:

$$P(0) - P(1) = \Re(\langle 0|^{\otimes n} V |0\rangle^{\otimes n}) = \Re(\langle 0|^{\otimes n} U^\dagger_{\phi(\vec{x_j})} U_{\phi(\vec{x_i})} |0\rangle^{\otimes n}),$$

tal y como queríamos demostrar.

4.3. Circuito Variacional

Hasta este punto, hemos visto como codificar los datos en el circuito cuántico y posteriormente calcular el producto escalar de dos estados distintos. Con todo esto, ya podríamos resolver (3.7) para obtener nuestro clasificador lineal. No obstante, el problema actual de esto es que se está suponiendo que el espacio objetivo de nuestro mapa de característica separa linealmente los datos siendo esto una suposición muy fuerte. Para evitar este tipo de suposiciones, se podría crear lo que se denomina circuito variacional. Un circuito de este tipo no es más que una serie de puertas que dependen de unos parametros $\vec{\theta}$ los cuales se irán adaptando en función de unos resultados obtenidos con el objetivo de crear el mapa característico que mejor se adapte a nuestro problema.

Juntando todas las piezas derivadas hasta este momento, llegamos a la expresión fundamental de nuestro Quantum Support Vector Machine variacional:

$$L = -\sum_{i=0}^{m} \alpha_i + \frac{1}{2}\sum_{i=0}^{m}\sum_{j=0}^{m} \alpha_i \alpha_j f(\vec{x_i}) f(\vec{x_j}) \Re(\langle 0|U^\dagger_{\phi(\vec{x_i},\vec{\theta})} U_{\phi(\vec{x_j},\vec{\theta})}|0\rangle) \tag{4.3}$$

siendo $U_{\phi(\vec{x_j},\vec{\theta})}$ nuestro nuevo mapa de características condicionado por θ. En este caso, los parametros que queremos optimizar serán tanto $\vec{\alpha}$ como $\vec{\theta}$ con el objetivo de minimizar L. Sin embargo, hay un factor importante que debemos tener en cuenta. Una de las características por las que se distingue el SVM sobre otros métodos de clasificación, es el hecho de que la función objetivo a minimizar cumple que es convexa y por tanto se convierte en una busqueda rápida y eficaz de un mínimo. Ahora bien, al introducir los parametros $\vec{\theta}$ estamos ganando flexibilidad en el kernel pero estamos rompiendo con esa convexividad por lo que se dificulta la resolución del proceso de optimización.

Por este motivo, se ha decidido realizar una doble optimización separando por un lado el proceso de ajuste de los parámetros $\vec{\alpha}$ de los $\vec{\theta}$ para aprovechar al máximo dicha propiedad de convexidad. En (4.3) se muestra el flujo de ejecución del algoritmo en cuestión.

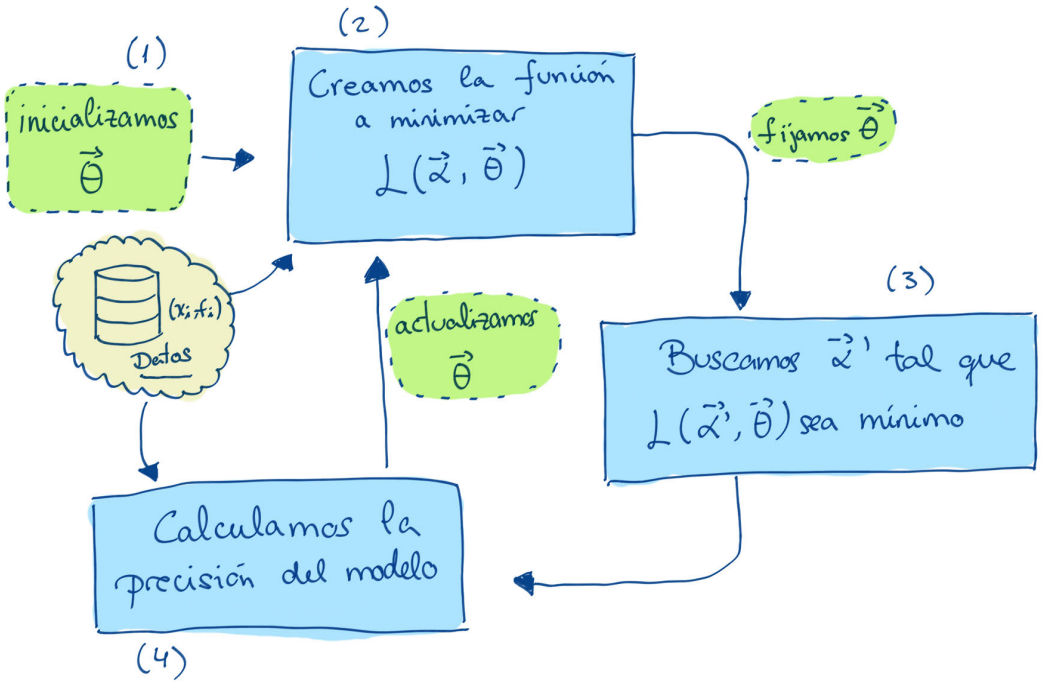


Figura 4.1: Diagrama de flujo de ejecución

El primero de los pasos tras la inicialización de $\vec{\theta}$ será construir la función (4.3). Después de esto, tras fijar la variable $\vec{\theta}$ podemos buscar el valor de $\vec{\alpha}$ que minimice L. Este proceso, como comentábamos anteriormente, se puede realizar rápidamente al ser ésta una función convexa. Dicha optimización puede realizarse aprovechando la diferenciabilidad de la función, con descenso del gradiente. Tras encontrar el valor buscado, tendríamos una primera aproximación al clasificador con la cual, a través del dataset podemos estimar un nivel de precisión. Este valor será el que se utilice para la actualización de la variable $\vec{\theta}$ que se llevará a cabo con un optimizador, en este caso, que no necesite gradiente como COBYLA [12]. Esta elección se debe a que el cálculo de la

precisión del modelo a través del dataset rompe la derivabilidad de la función[2]. Cabe destacar que este procedimiento es algo bastante novedoso tal y como se puede ver en trabajos recientes [5] (Mayo del 2021) donde se están desarrollando estas ideas.

4.4. El problema a tratar

Para poner a prueba nuestro algoritmo se ha trabajado con un dataset muy conocido en el mundo del Deep Learning, 'The Breast Cancer Dataset'. Con este conjunto de datos se pretende clasificar imágenes de diapositivas en dos clases. Imágenes que contienen células cancerígenas de mama y otras que no. Esta enfermedad, que afecta mayoritariamente a mujeres, tiene una elevada incidencia en nuestro país llegandose a diagnosticar el año pasado más de 30,000 nuevos casos[3]. Esta situación ha creado un marco de interés para aplicar conceptos de machine learning e inteligencia artificial en tareas de detección de este tipo de enfermedades en el que se han visto resultados muy prometedores en estos últimos años [14] [6]. Aquí podemos ver imágenes de lo que es el dataset original:

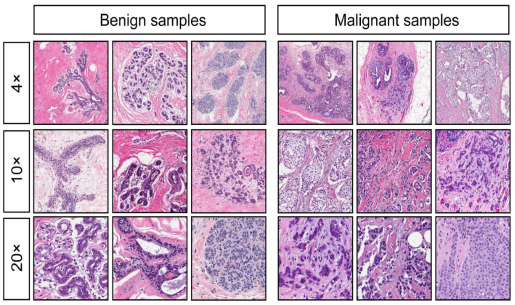


Figura 4.2: Imágenes correspondientes al Dataset Breast Cancer

No obstante, nosotros no trabajaremos con las imágenes sino con los atributos que podemos extraer de ellos. En concreto se tomarán 10 características distintas y de cada una de ellas se ofrecerá como atributo la media, la desviación estandar y el caso más desfavorable, es decir, serán un total de 30 atributos totales:

1) Radio (media de las distancias del centro a los puntos del perímetro)

2) Textura (desviación estandar de los valores en escala de grises)

3) Perímetro

[2]Esto se debe a que para clasificar un elemento hace falta evaluarlo a través la función $h(\vec{x}) = \vec{w}.\vec{x} + b$, y con dicho valor tendremos que aplicar la función signo para determinar si al elemento $\vec{x}$ se le asigna la etiqueta 1 o la etiquta -1. La función signo no es diferenciable

[3]Cifra proporcionada por la Sociedad Española de Oncología Médica (SEOM).

4) Área

5) Suavidad (variación local de las longitudes de los radios)

6) Compacidad (perímetro^2/área $-1{,}0$)

7) Concavidad (gravedad de las partes cóncavas del contorno)

8) Puntos cóncavos (número de porciones cóncavas del contorno)

9) Simetría

10) Dimensión fractal (.ªproximación a la línea de costa 1)

Por ejemplo, el atributo 1 será el radio medio, el campo 11 será la desviación típica de los radios y el campo 21, el menor de ellos.

Por jugar de una manera sencilla con el dataset hemos decidido en primer lugar realizar una reducción de la dimensión, es decir; utilizaremos las características más importantes para nuestro modelo (o una combinación de ellas).

4.4.1. Análisis de Componentes Principales

Para conseguir esa reducción de la dimensión existe una técnica muy usada denominada PCA o Análisis de Componentes Principales cuyo objetivo es describir el conjunto de datos en función de nuevas variables no correlacionadas[4]. Para su cálculo, el primer paso es centrar los datos, es decir, restar a cada punto la media total para que la media de los nuevos puntos esté situada en el origen. Después se trata de buscar el vector director $\vec{r}$ que maximice la siguiente expresión:

$$\max_{\vec{r}} r^t M_{\text{Cov}} \vec{r}, \tag{4.4}$$

siendo M_{Cov} la matriz de covarianza asociada a nuestros datos[4] y r^t la transpuesta del vector director $\vec{r}$. Desde un punto de vista más intuitivo se pretende buscar la recta que mantenga la mayor variabilidad de los datos posible.

Para entender el procedimiento mostremos un ejemplo sencillo en el que dispondremos de un conjunto de tres observaciones, las cuales estarán definidas con dos características: $A = [0, 0]$, $B = [3, 1]$ y $C = [0, 2]$. Sumando cada componente y dividiendo entre 3 obtenemos que el punto medio es $M = [1, 1]$, por lo que las nuevas variables centradas serán: $A' = [-1, -1]$, $B' = [2, 0]$ y $C' = [-1, 1]$.

El siguiente paso para continuar será calcular la matriz de covarianza M_{Cov}. Para ello recordemos que:

[4]En la matriz de covarianza, el término i,j hace referencia a la convarianza entre la i-ésima variable y la j-ésima.

$$\mathrm{Cov}(\vec{x},\vec{y}) = \frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y}) \tag{4.5}$$

Sustituyendo nuestras variables llegamos a:

$$M_{\mathrm{Cov}} = \begin{pmatrix} 2 & 0 \\ 0 & \frac{2}{3} \end{pmatrix}$$

En este caso en particular la matriz de covarianza es diagonal, simplificando mucho el proceso. El objetivo es buscar un $\vec{r}$ que maximice la expresión (4.4) por lo que buscaremos los parámetros α y β que maximicen la siguiente función:

$$g(\alpha,\beta) := \begin{pmatrix} \alpha & \beta \end{pmatrix} \begin{pmatrix} 2 & 0 \\ 0 & \frac{2}{3} \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \end{pmatrix}. \tag{4.6}$$

En particular, operando llegamos a:

$$g(\alpha,\beta) = 2\alpha^2 + \frac{2}{3}\beta^2,$$

y al estar sujeto a que $\alpha^2+\beta^2 = 1$ (por ser $\vec{r}$ unitario), obtenemos que $\alpha = 1$ y $\beta = 0$. Por consiguiente la recta que marcará nuestra primera componente principal será aquella centrada en $[1,1]$ y con vector director $[1,0]$.

No obstante, no debemos perder de vista el objetivo de esta técnica: reducir la dimensionalidad de nuestro dataset. Es por este motivo que una vez encontrada la recta en cuestión debemos hacer una proyección sobre la misma. Para ello, dado un punto $\vec{x}$ centrado, deberemos multiplicarlo por nuestro vector director r:

$$A'.\vec{r} = \begin{pmatrix} -1 \\ -1 \end{pmatrix} \begin{pmatrix} 1 & 0 \end{pmatrix} = -1. \tag{4.7}$$

Del mismo modo obtendríamos que el nuevo valor de B' en la recta es 2 y de C', -1. De aquí podemos observar que efectivamente hemos conseguido disminuir la dimensión del dataset.

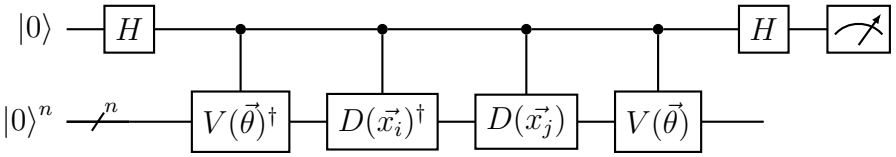
En concreto, de cara a nuestro dataset, se ha decidido hacer una reducción a dos componentes principales.

4.5. Resultados

Como comentábamos en la sección anterior, nuestro mapa de características es una aplicación $U_{\phi(\vec{x},\vec{\theta})}$ que actúa sobre un estado $|0\rangle$ por convenio. Sin embargo, por motivos de simplificación, es

frecuente suponer que las variables son independientes y por tanto se hace una factorización de $U_{\phi(\vec{x},\vec{\theta})}$ en dos bloques diferenciados. De este modo, podríamos definir $U_{\phi(\vec{x},\vec{\theta})}\,|0\rangle$ como $D(\vec{x})V(\vec{\theta})\,|0\rangle^n$ distinguiendo de esta manera entre una que depende del input y otra que depende de los parámetros variacionales.

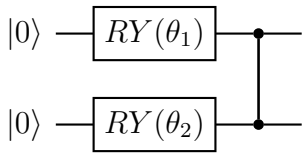
Por tanto el circuito a generar quedaría representado de la siguiente manera:



Una descomposición frecuente de la puerta V suele ser representada a partir puertas de rotación y puertas de control. En primer lugar definiremos la puerta $RY_k(\theta)$ como la aplicación de la puerta R_Y (2.2.3) al k-ésimo qubit. En segundo lugar el operador $CZ_{i,j}$ (Control-Z), será aquel que ejecuta una puerta Z sobre el j-ésimo qubit, si el i-ésimo qubit toma el valor $|1\rangle$. La definición general quedaría como:

$$V(\theta)\,|0\rangle^n = \prod_{k=0}^{n-2} CZ_{k,k+1} \prod_{k=0}^{n-1} RY_k(\theta)\,|0\rangle^n$$

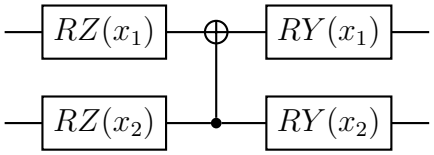
Como referencia para este tipo de circuitos encontramos trabajos como [5]. En particular nuestro circuito de dos qubits se construiría de la siguiente manera :



donde la recta que une ambos qubit define la puerta CZ. Por otro lado la descomposición en puertas cuánticas del operador D suele depender más del problema en cuestión. Nuestra primera aproximación será:

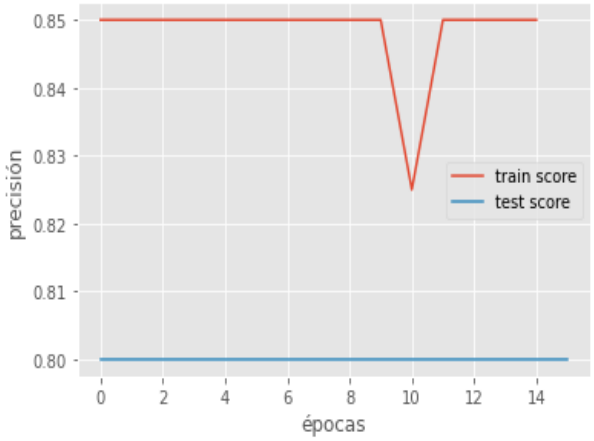
$$D(x) = \prod_{k=0}^{n-1} RY_k(\theta) \prod_{k=0}^{n-2} CX_{k,k+1} \prod_{k=0}^{n-1} RZ_k(\theta)$$

que del mismo modo, adaptado a nuestro circuito de dos qubits tendríamos:



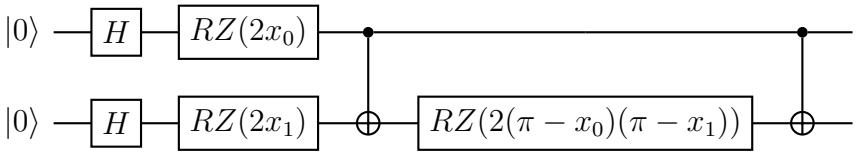
A continuación mostraremos una serie de gráficas en las que estudiaremos la precisión[5] del modelo a lo largo de una serie de *épocas*. Las *épocas* harán referencia a la evolución de nuestro modelo para cada nuevo ajuste de $\vec{\theta}$. Por este motivo, es esperable que para *épocas* más avanzadas, la precisión sea más alta ya que nuestro parámetro $\vec{\theta}$ se ha ido optimizando en cada nueva *época*.

Tras realizar la prueba de nuestro dataset con el mapa de características anteriormente definido, obtenemos un valor de precisión del 85 % para el conjunto de entrenamiento y del 80 % para el conjunto test.



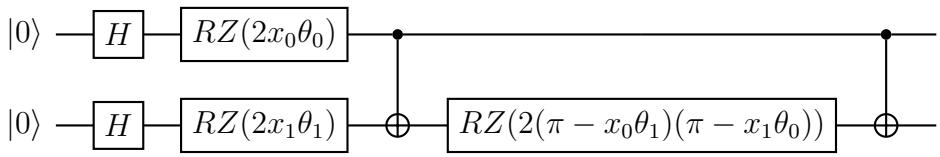
No obstante, aunque el resultado no es malo, vemos que a lo largo de las *épocas* el modelo no está mejorando su precisión a la hora de clasificar. Este hecho viene a representar que el parámetro $\vec{\theta}$ no juega ningún papel relevante en este proceso, por lo que se decidió descartar la suposición de la independencia de variables y buscar algún otro tipo de mapa de características.

En particular, uno muy usado en la comunidad de Qiskit[9] es el ZZ-FeatureMap [13] cuya construcción para circuitos de tamaño 2 quedaría definido de la siguiente manera:



Ahora bien, a diferencia del primer planteamiento realizado en esta sección, no haremos una separación en dos bloques para los parámetros $\vec{\theta}$ y los datos de entrenamiento $\vec{x}$, ya que consideramos que de esta manera los parámetros no tienen fuerza sobre el modelo en sí. Es por ello que definiremos nuestro propio mapa de características como:

[5]Entenderemos como precisión el número de aciertos partido por el número total de muestras.



El nuevo mapa de características transformado de esta manera nos permite crear un modelo flexible capaz de aprender y adaptarse a los datos en cuestión.

Continuaremos explicando unos bloques de código que hemos implementado para el cálculo del producto interno.

```
14 @qml.template
15 def feature_map(x,w):
16     # puerta V
17 
18     for _ in range(2):
19         for i in range(2):
20             qml.Hadamard(wires = i)
21 
22         qml.RZ(2*x[0]*w[0], wires = 0)
23         qml.RZ(2*x[1]*w[1], wires = 1)
24         qml.CNOT(wires = [0,1])
25         qml.RZ(2*(np.pi - x[0]*w[1])*(np.pi - x[1]*w[0]), wires = 1)
26         qml.CNOT(wires = [0,1])
27
```

En esta primera parte estamos creando lo que sería el mapa de características tal cual se definió anteriormente, donde x es el valor correspondiente al dataset y w (equivalente a nuestro $\vec{\theta}$) hace referencia al parámetro de entrenamiento de dicho mapa. Por tanto será esa variable w la que se irá ajustando en función de la precisión que vayamos obteniendo del modelo.

```
35 @qml.template
36 def circuit(x,y,w):
37     feature_map(y,w)
38     qml.inv(feature_map(x,w))
39 
40 dev = qml.device("default.qubit", wires = n + 1)
41 @qml.qnode(dev)
42 def h_test(x,y,w):
43         ops = qml.ctrl(circuit, control = n)
44         qml.Hadamard(n)
45         ops(x = x, y = y, w = w)
46         qml.Hadamard(n)
47         return qml.probs(wires = n)
48 
49 def scalar_product(x,y,w):
50     probs = h_test(x,y,w)
51     return probs[0] - probs[1]
52
```

En este segundo bloque hemos definido el circuito al que aplicaremos el Hadamard Test correspondiente con la concatenación del mapa de características evaluado en un punto y y la inversa[6] de éste aplicado a un punto x. Finalmente obtendremos el producto escalar deseado como la probabilidad de que el primer qubit sea 0 menos la probabilidad de que tome el valor 1.

[6]En la línea 38 podemos ver el método para calcular la inversa. No obstante, con la nueva actualización de Pennylane caerá en desuso por 'qml.adjoint'.

En la gráfica siguiente se muestra la precisión obtenida a lo largo de 15 *épocas* con este nuevo planteamiento. Como se puede apreciar, tras realizar el entrenamiento con este modelo hemos

Figura 4.3: Precisión del modelo según el número de *épocas* para el dataset de entrenamiento y prueba.

obtenido una precisión inicial del 50 % con el dataset de entrenamiento y un 50 % con el de test, y después de realizar el ajuste de $\vec{\theta}$ se ha llegado al 95.5 % y 90 % respectivamente.

En este tipo de optimizaciones es normal no ver una mejora progresiva a lo largo de las *épocas* como podría ocurrir por ejemplo con el uso de redes neuronales. Esto se debe a que no estamos utilizando un procedimiento de mejora a través de descenso de gradiente, el algoritmo utilizado está realizando una busqueda a ciegas que irá afinando a lo largo del tiempo.

4.6. Discusión

A lo largo del proyecto hemos ido desarrollando este algoritmo híbrido, y aunque se ha ido definiendo poco a poco, veo conveniente hacer un resumen sobre las partes que se ejecutan en computadores clásicos y cuánticos. La primera de las etapas fue la reducción de dimensionalidad, realizada a través del PCA de manera clásica. Después de esto se tuvo que calcular la matriz de Gram de nuestro dataset, llevado a cabo con computadores cuánticos. Una vez obtenidos los valores de dicha matriz, se buscó el hiperplano de clasificación minimizando con ordenadores clásicos nuestra función objetivo y, finalmente, para determinar la clase de un nuevo dato, se calculó el producto interno con respecto a los datos de entrenamiento en el computador cuántico.

Por otro lado, nuestro kernel final utilizado estaba formado por dos qubits y siete puertas cuánticas, y por tanto, si quisiéramos hacer una adaptación directa de este kernel en un computador clásico, tendríamos que multiplicar 7 matrices de dimensión $2^2 \times 2^2$ en total[7]. Puede que parezca que no

[7]Recordemos que una puerta cuántica se puede simular en un ordenador clásico como una matriz de $2^n \times 2^n$ siendo n el número de qubits del circuito.

supone mucho más coste pero el tamaño de estas matrices crece de forma exponencial con relación al número de qubits.

Con el dataset utilizado, se ha visto que el SVM clásico también es capaz de conseguir buenos resultados pero a través del uso de otro tipo de kernels. El reto actual está en identificar datasets en los que sea complicado encontrar núcleos alternativos.

Capítulo 5

Desarrollo y coste del proyecto

El desarrollo del proyecto se ha llevado a cabo a lo largo de 6 meses en los que se ha podido dedicar diariamente dos horas. Esto ha dado lugar a un total de 240h trabajadas entre las que podemos incluir las reuniones de control semanales que se han llevado a cabo con IBM. A continuación intentaremos realizar una aproximación a los costos de este proyecto. Para ello deberemos ver cuales han sido los diferentes roles que hemos tomado a lo largo de éste, así como las horas dedicadas.

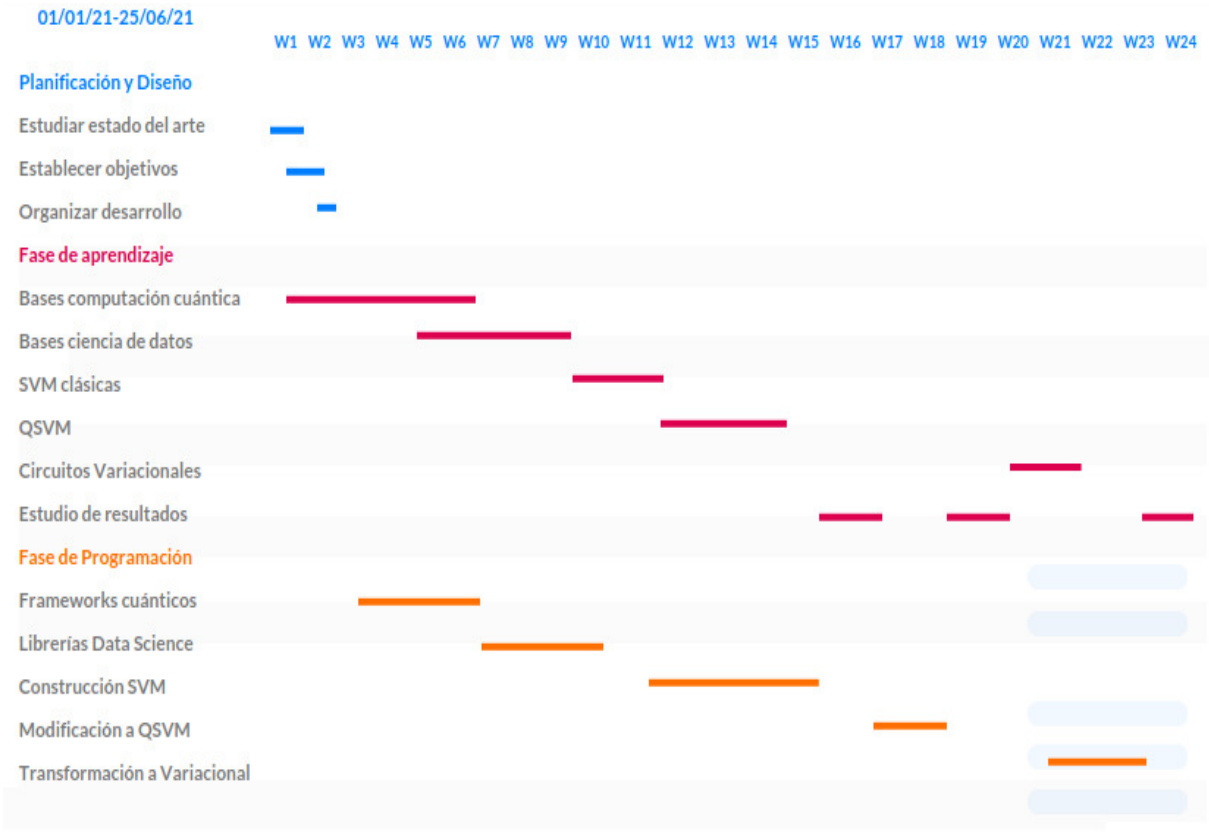
Roles	Horas dedicadas	€/ Hora	€totales
Programador	83h	30.00€	2490.00€
Investigador Principal	119h	30.00€	3570.00€
Jefe de Proyectos	38h	35.50€	1349.00€

Los salarios medios se han tomado actualizados en el año 2021 ajustados con unas estimaciones ofrecidas por IBM. El rol de programador tiene un salario superior a la media ya que requiere de una mayor especialización para el desarrollo cuántico. Se ha llevado a cabo un estudio dicho salario observando las empresas principales del sector como pueden ser IBM, Xanadu o Zapata.

Esto hace un total de 7409.00€. No obstante, este no es el presupuesto del proyecto sino que deberemos hacer un incremento para cubrir otro tipo de gastos como podría ser la seguridad social. En líneas generales, este incremento a la hora de estimar suele hacerse de un 40 % dejando nuestra estimación final en 10372.60€.

Dichos roles no se ejercieron de forma uniforme a lo largo del proyecto, ya que en una primera etapa el mayor peso se lo llevó el trabajo de investigación. Este tema al ser tan actual es difícil encontrar documentación y el esfuerzo de generarla ha sido significativo. La segunda etapa fue llevada a cabo a través del rol de programador. El código fue desarrollado a través del framework de Xanadu orientado al Quantum Machine Learning. El peso de la etapa final se lo llevó el desarrollo del proyecto en sí: organización y ejecución del documento actual, busqueda de datasets de interés, pruebas, etc. Todas las etapas fueron acompañadas del rol de Gestor de Proyectos, necesario para

enfocar adecuadamente los objetivos y garantizar el correcto avance del mismo.



Como podemos ver en el diagrama de Gantt generado, el mayor peso de trabajo a lo largo de estos meses se lo ha llevado la fase de aprendizaje, ya que como ha quedado constancia, el proyecto realizado ha sido principalmente teórico.

Capítulo 6

Conclusión y trabajos futuros

A lo largo de este periodo de trabajo hemos podido hacer un primer acercamiento a la rama del aprendizaje automático cuántico. Aunque el sector está naciendo ahora mismo, el hecho de poder realizar ejecuciones en computadores cuánticos reales supone un avance muy notable. Por otro lado, la inteligencia artificial ha traído con sigo un cambio revolucionario y siendo su mayor limitación la eficiencia en ciertos algoritmos, la computación cuántica parece una gran tecnología a tener en cuenta para solucionar este tipo de problemas.

Es por este motivo que consideramos este primer acercamiento realizado a través de la computación híbrida muy acertado a la vez que didáctico. En futuros trabajos resultaría muy interesante poder llevar a cabo un estudio mayor acerca de los datasets en los que la computación cuántica podría marcar diferencias más significativas respecto a la programación tradicional ya que a día de hoy sigue siendo un problema abierto de gran interés.

Por último agradecer de nuevo a Javier Machín y Ginés Carrascal su continuo apoyo a lo largo de todo el proyecto representando a IBM, su seguimiento semanal y el buen trato personal han facilitado la ejecución del proyecto.

Bibliografía

[1] Ville Bergholm y col. *PennyLane: Automatic differentiation of hybrid quantum-classical computations*. 2020. arXiv: 1811.04968 [quant-ph].

[2] Jozsa Richard Deutsch David. "Rapid solution of problems by quantum computation". En: 4 (1992). URL: http://doi.org/10.1098/rspa.1992.0167.

[3] Theodoros Evgeniou y Massimiliano Pontil. "Support Vector Machines: Theory and Applications". En: vol. 2049. Ene. de 2001, págs. 249-257. DOI: 10.1007/3-540-44673-7_12.

[4] Felipe L. Gewers y col. "Principal Component Analysis". En: *ACM Computing Surveys* 54.4 (mayo de 2021), págs. 1-34. ISSN: 1557-7341. DOI: 10.1145/3447755. URL: http://dx.doi.org/10.1145/3447755.

[5] Jennifer R. Glick y col. *Covariant quantum kernels for data with group structure*. 2021. arXiv: 2105.03406 [quant-ph].

[6] Simon Hadush y col. *Breast Cancer Detection Using Convolutional Neural Networks*. 2020. arXiv: 2003.07911 [cs.CV].

[7] Vojtěch Havlíček y col. "Supervised learning with quantum-enhanced feature spaces". En: *Nature* 567.7747 (mar. de 2019), págs. 209-212. ISSN: 1476-4687. DOI: 10.1038/s41586-019-0980-2. URL: http://dx.doi.org/10.1038/s41586-019-0980-2.

[8] Jamie Heredge y col. *Quantum Support Vector Machines for Continuum Suppression in B Meson Decays*. 2021. arXiv: 2103.12257 [quant-ph].

[9] Daniel Koch, Laura Wessing y Paul M. Alsing. *Introduction to Coding Quantum Algorithms: A Tutorial Series Using Qiskit*. 2019. arXiv: 1903.04359 [quant-ph].

[10] Michael A. Nielsen e Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[11] Jae-Eun Park y col. *Practical application improvement to Quantum SVM: theory to practice*. 2020. arXiv: 2012.07725 [quant-ph].

[12] M. Powell. "A View of Algorithms for Optimization Without Derivatives". En: *Mathematics TODAY* 43 (ene. de 2007).

[13] Yudai Suzuki y col. “Analysis and synthesis of feature map for kernel-based quantum classifier”. En: *Quantum Machine Intelligence* 2.1 (jun. de 2020). ISSN: 2524-4914. DOI: 10.1007/s42484-020-00020-y. URL: http://dx.doi.org/10.1007/s42484-020-00020-y.

[14] Juan Zuluaga-Gomez y col. *A CNN-based methodology for breast cancer diagnosis using thermal images*. 2019. arXiv: 1910.13757 [cs.CV].