



MÁQUINAS DE VECTOR SOPORTE CUÁNTICO: ALGORITMO DE *MACHINE LEARNING* CUÁNTICO

AUTORES

Alejandro Delgado Ontana
Javier Alexander Maza Valdevieso

ÍNDICE

1.	INTRODUCCIÓN.....	3
2.	MÁQUINAS DE VECTOR SOPORTE	3
2.1.	<i>FUNCIONAMIENTO</i>	3
2.2.	<i>OBTENCIÓN DE VECTORES SOPORTE</i>	4
2.3.	<i>KERNELS</i>	6
3.	QSVM	7
3.1.	<i>MAPA DE CARACTERÍSTICAS</i>	7
3.2.	<i>PRODUCTO ESCALAR</i>	8
3.3.	<i>CIRCUITO VARIACIONAL</i>	8
4.	IMPLEMENTACIÓN DE UNA QSVM	10
5.	CONCLUSIONES	11
6.	BIBLIOGRAFÍA	12

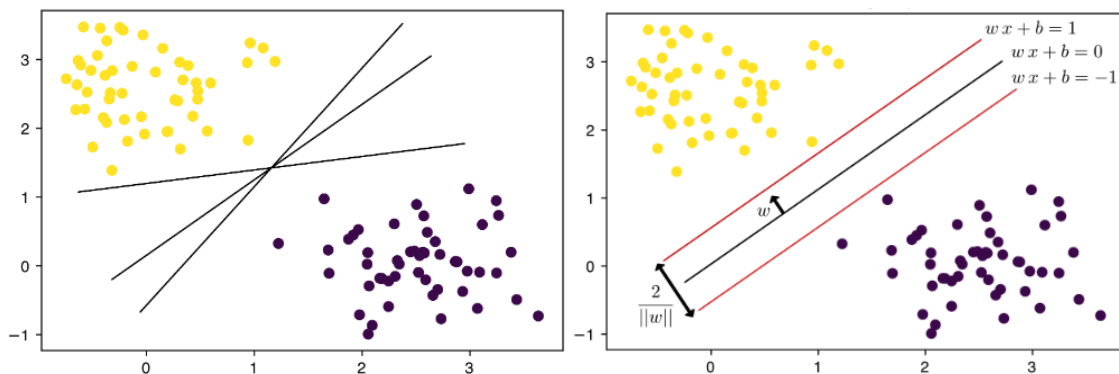
1. INTRODUCCIÓN

Uno de los problemas más comunes, y complejos a los que se ha enfrentado el ser humano son los problemas de clasificación. 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 podemos destacar las máquinas de vector soporte (SVM). Dichos algoritmos se pueden dividir en dos clases: aprendizaje supervisado, en el que se dispone de un conjunto de datos de entrada etiquetados; y no supervisado, en los que no tenemos información previa de cómo clasificar los 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 la computación cuántica.

2. MÁQUINAS DE VECTOR SOPORTE

2.1. FUNCIONAMIENTO

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-. Supondremos que tenemos únicamente dos clases separables. Como podemos ver en la imagen, existen infinitud de posibles planos que nos servirían para separar nuestros conjuntos, sin embargo, para las SVMs no es suficiente con encontrar un plano que cumpla esta condición. La meta real es encontrar el hiperplano que maximice las distancias entre las dos clases.



Ejemplo de funcionamiento de una SVM

La distancia entre el hiperplano y los puntos más cercanos de ambas clases recibe el nombre de *margen*. Dichos puntos más próximos al hiperplano, se conocen como *puntos soporte* y son los que realmente determinan la posición final de éste. El algoritmo detrás de las SVM proporciona un método para encontrar los puntos críticos y obtener el hiperplano solución.

2.2. OBTENCIÓN DE VECTORES SOPORTE

Sea $H = \left\{ \vec{x} \in \frac{\mathbb{R}^n}{\vec{x}} \vec{w} + b = 0 \right\}$ el hiperplano buscado definido por $\vec{w} \in \mathbb{R}^n$ y $b \in \mathbb{R}$. Definimos 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 $\vec{x} \in \Omega$. En este caso, al trabajar únicamente con dos clases, diremos que $f(\vec{x}) \in \{-1, 1\}$. De esta forma sería suficiente encontrar unos parámetros \vec{w} y b que se cumplan:

$$f(\vec{x})(\vec{w}\vec{x} + b) \geq 0$$

Aunque lo que queremos maximizar es la longitud del *margen*. Por ello, definimos los planos que lo limitan

$$H_1 = \left\{ \vec{z} \in \frac{\mathbb{R}^n}{\vec{z}} \vec{w} + b = 1 \right\}, H_2 = \left\{ \vec{z} \in \frac{\mathbb{R}^n}{\vec{z}} \vec{w} + b = -1 \right\}$$

Y el plano que separa los *márgenes* por la mitad

$$H = \left\{ \vec{z} \in \frac{\mathbb{R}^n}{\vec{z}} \vec{w} + b = 0 \right\}$$

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 H_2 y H . Se puede definir \vec{z} de esta manera porque 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 queda:

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

Sabiendo que $\vec{y} \in H_2$ y $\vec{z} \in H$ se tiene que:

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

llegamos a que la distancia entre H_2 y H es $\frac{1}{\|\vec{w}\|}$. Por simetría, la distancia entre H_1 y H será igual, luego el problema será minimizar $\frac{\|\vec{w}\|^2}{2}$ sujeto a:

$$f(\vec{x}_i)(\vec{w}\vec{x}_i + b) \geq 1, \forall \vec{x}_i \in \Omega$$

ya que buscando maximizar la distancia entre planos, garantizamos 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). Luego, en base a la función objetivo y las restricciones definidas, el objetivo final es encontrar un mínimo en la siguiente función:

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

siendo 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:

$$\begin{aligned} \frac{\partial L}{\partial \vec{w}} &= \vec{w} + \sum_{i=0}^m \alpha_i f(\vec{x}_i) \vec{x}_i \\ \frac{\partial L}{\partial b} &= \sum_{i=0}^m \alpha_i f(\vec{x}_i) \end{aligned}$$

Conseguimos, sustituyendo en L , llegar al problema dual:

$$L' = - \sum_{i=0}^m \alpha_i + \frac{1}{2} \sum_{i=0}^m \sum_{j=0}^m \alpha_i \alpha_j f(\vec{x}_i) f(\vec{x}_j) \vec{x}_i \cdot \vec{x}_j$$

dándonos cuenta de que $\sum_{i=0}^m \alpha_i f(\vec{x}_i) \vec{w} \vec{x}_i$ es constante por ser equivalente a $\vec{w} \vec{w}$. Nótese que al ser las restricciones desigualdades, siguiendo la formulación KKT, en este caso, se añade que se debe cumplir la siguiente condición $\alpha_i [f(\vec{x}_i)(\vec{w} \vec{x}_i + b) - 1] = 0$. Podemos

deducir que aquellos puntos no pertenecientes al plano soporte harán que α_i valga 0. Una vez hallados los valores α_i procedemos a recuperar \vec{w} y b .

Podemos calcular fácilmente \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, se cumple que $f(\vec{x}_i)(\vec{w}\vec{x}_i + b) - 1 = 0$. Multiplicando a ambos lados por $f(\vec{x}_i)$ y despejando, obtenemos $b = f(\vec{x}_i) - \vec{w}\vec{x}_i$.

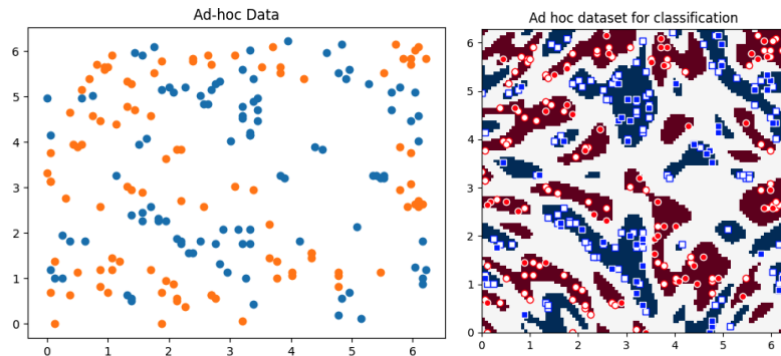
Podríamos tomar b , entonces, de la siguiente manera:

$$b = \frac{1}{S} \sum_{i=1}^S (f(\vec{x}_i) - \vec{w} \cdot \vec{x}_i)$$

donde S es el número de vectores soporte y los \vec{x}_i dichos vectores. Esta formulación será de gran ayuda más adelante.

2.3. KERNELS

Para todo lo explicado hasta ahora hemos tenido en cuenta una suposición muy fuerte: los datos eran linealmente separables. A través de los *kernels* podemos lidiar con situaciones en las que esto no se dé.



Representación de espacio con datos no separables linealmente

Definimos ϕ como la función kernel que mapea cada elemento en uno de dimensión mayor, entonces, obtenemos el mismo problema salvo con una pequeña modificación:

$$L = -\sum_{i=0}^m \alpha_i + \frac{1}{2} \sum_{i=0}^m \sum_{j=0}^m \alpha_i \alpha_j f(\vec{x}_i) f(\vec{x}_j) \phi(\vec{x}_i) \cdot \phi(\vec{x}_j)$$

y, por simplificación, podemos definir la función kernel como $K(\vec{x}_i, \vec{x}_j) := \phi(\vec{x}_i)\phi(\vec{x}_j)$. Nótese que visto de esta manera no nos hace falta calcular explícitamente los valores $\phi(\vec{x}_i)$, sino que solo será necesario obtener el producto interno de dos elementos transformados por el mapeo ϕ . Ahora bien, a la hora de obtener el valor \vec{w} sí debiéramos calcular todos los valores de la función *kernel*:

$$\vec{w} = - \sum_{i=0}^m \alpha_i f(\vec{x}_i) \phi(\vec{x}_i)$$

pero esto es algo que podemos evitar teniendo en cuenta que nuestro objetivo final es clasificar, no calcular \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 función del resultado el elemento correspondiente pertenecería a una clase u otra.

El procedimiento es este punto suele ser precalcular todos los productos $K(\vec{x}_i, \vec{x}_j)$ $\forall i, j \in \{1, \dots, n\}$. 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 $K(\vec{x}_i, \vec{x}_j)$.

3. QSVM

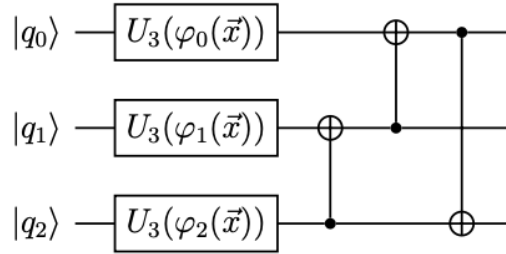
El aprendizaje automático cuántico (*Quantum Machine Learning*) es una área de investigación emergente que podemos situar entre la mecánica cuántica y la informática. Busca aprovechar el paralelismo intrínseco de la computación cuántica para proporcionar soluciones al análisis de grandes volúmenes de datos. Lo que estudiaremos a continuación será una forma de adaptar el algoritmo clásico de *Machine Learning* de aprendizaje supervisado SVM a nivel cuántico.

3.1. MAPA DE CARACTERÍSTICAS

El primer paso para construir el clasificador QSVM consiste en mapear los datos $\vec{x} \in \Omega$ en un circuito cuántico. Este es un procedimiento que se lleva a cabo a través del mapa de características definido por una puerta $U_{\phi(\vec{x})}$ siendo $\phi: \vec{x} \rightarrow \mathbb{R}^n$ una función *kernel* asociada. Dicha puerta se asocia a un estado inicial, por lo que la representación cuántica del vector \vec{x} en el circuito será $U_{\phi(\vec{x})}|0\rangle^{\otimes n}$. A continuación, mostraremos algún mapa 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 $\varphi: \vec{x} \rightarrow (0, 2\pi] \times (0, 2\pi] \times [0, \pi]$, por lo que el mapa de características quedaría definido como $\vec{x} \rightarrow |\phi_i(\vec{x})\rangle = U(\varphi_i(\vec{x}))|0\rangle$,

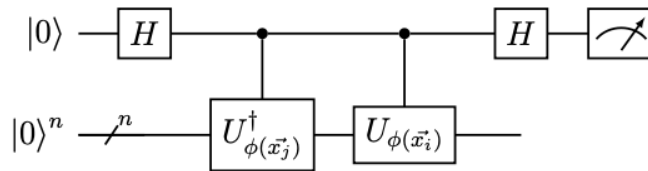
$\forall i \in 1, 2, \dots, n$. Además, es frecuente que los mapas jueguen con el entrelazamiento para aprovechar el potencial de la computación cuántica. Una forma sencilla de añadir esto al circuito sería a través de una serie de puertas *CNOT*. Un ejemplo de mapa de características que implementa lo mencionado sería:



Ejemplo de mapa de características

3.2. PRODUCTO ESCALAR

Tal cual está definida la ecuación L -ya mencionada- del algoritmo, no hace falta el estado en sí para llegar al mínimo sino el producto escalar de la imagen de dos estados iniciales: dados \vec{x}_i, \vec{x}_j se busca el valor $\langle 0 |^{\otimes n} U_{\phi(\vec{x}_j)}^t U_{\phi(\vec{x}_i)} | 0 \rangle^{\otimes n}$ siendo U^t la matriz traspuesta del conjugado complejo de U . Sin embargo, este valor podría ser complejo y nos interesaría que tomase valores reales, por lo que sólo obtendremos la parte real de dicho valor. Para calcularlo nos ayudaremos de lo que se conoce como *Hadamard Test*, cuya estructura es la siguiente:



Representación Test de Hadamard

De esta forma, se tendría que $\Re \left(\langle 0 |^{\otimes n} U_{\phi(\vec{x}_j)}^t U_{\phi(\vec{x}_i)} | 0 \rangle^{\otimes n} \right) = P(1) - P(0)$ siendo $P(0)$, $P(1)$ la probabilidad de obtener $|0\rangle$ y $|1\rangle$ respectivamente, si medimos el primer qubit.

3.3. CIRCUITO VARIACIONAL

Como ya hemos comentado, un gran error a la hora de intentar obtener la solución del problema con las herramientas mencionadas es suponer que el espacio objetivo de nuestro mapa de características separa linealmente los datos. Para no llevar a cabo este tipo de suposiciones tan fuertes, se puede recurrir a los circuitos variacionales. Un circuito de este tipo no es más que una serie de puertas que dependen de unos

parámetros $\vec{\theta}$ que se ajustan en función a ciertos resultados obtenidos, con el objetivo de crear un mapa que se adapte al problema tratado.

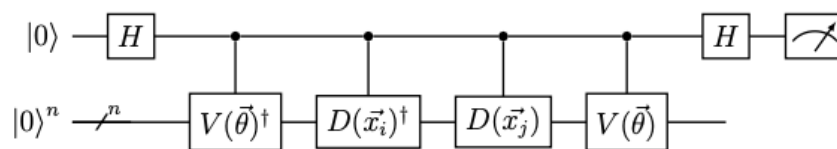
Juntando todas las piezas mencionadas hasta ahora, llegamos a la expresión fundamental de un QSVM variacional:

$$L = - \sum_{i=0}^m \alpha_i + \frac{1}{2} \sum_{i=0}^m \sum_{j=0}^m \alpha_i \alpha_j f(\vec{x}_i) f(\vec{x}_j) \Re(\langle 0 | U_{\phi(\vec{x}_i, \vec{\theta})}^\dagger U_{\phi(\vec{x}_j, \vec{\theta})} | 0 \rangle)$$

Por otro lado, al introducir los parámetros $\vec{\theta}$ estamos ganando flexibilidad en el *kernel* pero estamos rompiendo la convexidad de la que goza la función objetivo y que hace tan factible la búsqueda de un mínimo. Por tanto, se decide realizar una doble optimización separando el proceso de ajuste de los parámetros $\vec{\alpha}$ y $\vec{\theta}$.

En primer lugar, tras fijar la variable $\vec{\theta}$, podemos buscar el valor de $\vec{\alpha}$ que minimice la función, operación relativamente sencilla por ser ésta convexa. Esto se puede realizar aprovechando la diferenciabilidad de la función, con descenso de gradiente. Tras encontrar el valor, tendríamos una primera aproximación al clasificador con lo cual, a través del *dataset* utilizado, podemos estimar un nivel de precisión. Este valor será el que se utilice para actualizar $\vec{\theta}$ que se llevará a cabo con un optimizador, normalmente que no utilice gradiente ya que el cálculo de la precisión del modelo a través del *dataset* rompe la derivabilidad de la función. Esto se debe a que para clasificar un elemento hace falta evaluarlo a través de la función $h(\vec{x}) = \vec{w}\vec{x} + b$ y a dicho valor le tendremos que aplicar la función signo para determinar qué etiqueta se le asigna a \vec{x} . Pero la función signo no es diferenciable.

Por último, como ya se ha explicado anteriormente, el mapa de características es una aplicación $U_{\phi(\vec{x}, \vec{\theta})}$ que actúa sobre un estado inicial $|0\rangle$. Sin embargo, para simplificar, es frecuente suponer que las variables son independientes, por lo que se hace una factorización de la aplicación en dos bloques diferenciados. Así, se tendría que $U_{\phi(\vec{x}, \vec{\theta})}$ se convierte en $D(\vec{x})V(\vec{\theta})|0\rangle^n$ y el circuito a generar quedaría de la siguiente forma:



Representación circuito desdoblado

Una descomposición frecuente de la puerta V suele ser representada a partir de la puerta $RY_k(\theta)$ como la aplicación de la puerta R_Y al k -ésimo qubit, y el operador $CZ_{i,j}$ que ejecuta una puerta Z sobre el j -ésimo qubit solo si el i -ésimo qubit toma el valor $|1\rangle$. Del mismo modo se descompone el operador D en puertas lógicas de rotación y control que dependerán más del problema en cuestión que se esté proponiendo.

4. IMPLEMENTACIÓN DE UNA QSVM

Una vez desarrollado todo el funcionamiento de una QSVM a nivel teórico, se quiere desarrollar un programa de ejemplo que implemente el algoritmo. En este caso práctico, utilizaremos el conocido *Cleveland Heart Disease dataset* que recoge la presencia de enfermedades cardíacas dependiendo de ciertos atributos como la edad, el sexo o algunos síntomas que puede padecer un paciente diagnosticado. Utilizaremos sólo cinco de los catorce atributos de los que se disponen en el *dataset* por simplificar el problema y disminuir el tiempo que toma el programa para realizar el entrenamiento.

```
data = pd.read_csv('Cleveland Dataset.csv')

X = data[['ca', 'cp', 'thal', 'exang', 'slope']]
y = data['target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

Después de leer el *dataset*, elegimos los atributos que queremos medir y separamos los datos diferenciándolos entre los que son de entrenamiento y los que se utilizarán para realizar los *tests* a la máquina entrenada.

```
samples = np.append(X_train, X_test, axis=0)
minmax_scaler = MinMaxScaler((0, 1)).fit(samples)
X_train = minmax_scaler.transform(X_train)
X_test = minmax_scaler.transform(X_test)
```

Obtenemos las etiquetas y las clases que definirán a qué atributo pertenecen los datos.

```
# number of qubits is equal to the number of features
num_qubits = 5
# regularization parameter
C = 1000

algorithm_globals.random_seed = 12345

backend = QuantumInstance(
    BasicAer.get_backend("qasm_simulator"),
    seed_simulator=algorithm_globals.random_seed,
    seed_transpiler=algorithm_globals.random_seed,
)
feature_map = ZFeatureMap(feature_dimension=num_qubits, reps=2)
qkernel = QuantumKernel(feature_map=feature_map, quantum_instance=backend)
qsvc = QSVC(quantum_kernel=qkernel, C=C)
```

Definimos el número de qubits a utilizar que coincidirá con el número de atributos escogido, la semilla utilizada por el computador cuántico y un parámetro regularizador. A continuación, nos conectamos al simulador de circuitos cuánticos (*qasm_simulator*) especificando su nombre y la semilla definida antes.

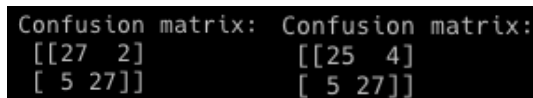
En este ejemplo, se ha utilizado un mapa de características predefinido por la biblioteca *Qiskit* -biblioteca utilizada durante toda la implementación- llamado *Z-FeatureMap*.

Con el mapa de características podemos construir el *kernel* que vamos a utilizar en el modelo y, por último, declarar el algoritmo que vamos a utilizar -en nuestro caso QSVM-.

```
# training
qsvc.fit(X_train, y_train)

# testing
qsvc_score = qsvc.score(X_test, y_test)
```

Entrenamos la máquina y hacemos las pruebas necesarias para saber si el entrenamiento ha sido efectivo. Al imprimir el valor *qsvc_score* sabremos que el porcentaje de acierto ha sido del 88.52%, y al ejecutar el mismo procedimiento utilizando una SVM clásica obtenemos un acierto del 85.24%. Las matrices de confusión de ambos casos se muestran a continuación:



Confusion matrix:	Confusion matrix:
[[27 2]	[[25 4]
[5 27]]	[5 27]]

IZQ: Matriz de confusión al aplicar QSVM
DCHA: Matriz de confusión al aplicar SVM

Cabe destacar el uso de dos bibliotecas básicas para la ejecución de estos casos de ejemplo:

- *Qiskit* -ya mencionada- que proporciona el simulador, el kernel o el mapa de características.
- *SciKit Learn* que proporciona las funcionalidades para el procesamiento de los datos y los resultados.

Además, se han usado bibliotecas características de Python como *numpy* o *pandas* para la lectura del *dataset*.

5. CONCLUSIONES

Con el *dataset* utilizado en este ejemplo, se puede ver que el SVM clásico obtiene buena precisión, pero, a través de otro tipo de *kernels* y empleando los conocimientos en computación cuántica que se tienen hoy en día, se pueden lograr mejores resultados y una mayor precisión. El reto actual está en identificar *datasets* en los que sea complicado encontrar núcleos alternativos para que juegue, aún más si cabe, un papel más importante el uso de la computación cuántica.

Por otro lado, es importante destacar el poder computacional de utilizar métodos y algoritmos cuánticos en la resolución de este tipo de problemas de clasificación, pues si quisiéramos hacer una adaptación directa de un *kernel* cuántico en un computador

clásico, tendríamos que multiplicar tantas matrices $2^n \times 2^n$ como puertas tenga el circuito cuántico, siendo n el número de qubits del circuito.

6. BIBLIOGRAFÍA

- Daniel Koch, Laura Wessing y Paul M. Alsing. *Introduction to Coding Quantum Algorithms: A Tutorial Series Using Qiskit*. 2019.
- Jae-Eun Park y col. *Practical application improvement to Quantum SVM: theory to practice*. 2020 .
- Qiskit documentation: <https://qiskit.org/documentation/>.
- Theodoros Evgeniou y Massimiliano Pontil. *Support Vector Machines: Theory and Applications*. 2001.
- SciKit Learn: <https://scikit-learn.org/stable/>.
- Amit Patidar. *How to implement QSVM in IBM Q environment*: <https://www.sigmoid.com/blogs/quantum-computing-blog-3-how-to-implement-qsvm-in-the-ibm-q-environment/>.
- Guillermo Alonso Alonso de Linaje. *Estudio de QSVM, algoritmo de Machine Learning cuántico*. 2021.