Manual Quantum Support Vector Machines

Luis Cervantes Guevara
Asesor: Juan Mauricio Torres González

Octubre 2021

Índice general

Resumen

Con el objetivo de mostrar formas de utilizar herramientas dek computo cuántico en el proceso de clasificación de datos en maquinas de soporte vectorial (MSV) y ver como se comparan con la forma clásica en la que se hace esta tarea. Partimos desde la definición de una MSV mostrando su funcionamiento en el caso más básico, esto es con datos linealmente separables y como para resolver problemas más complejos introducimos técnicas tales como los espacios de características, funciones y matrices kernel. Después describimos maneras de sustituir ... aplicar circuitos cuánticos al proceso de clasificación de manera total o parcial, y finalmente .... un código donde se muestra la implementación de dichos circuitos con un set de datos y la comparación con un algoritmo clásico

0.1. Introducción: Máquinas de Soporte Vectorial

0.1.1. Clasificador de Soporte Vectorial

Las máquinas de soporte vectorial (SVM's, por sus siglas en inglés) son un método de clasificación de datos desarrollado por Vladimir Vapnik (1936) y su equipo en la década de 1990 [1].
Para entender cuál es el objetivo de este método, empecemos planteando el caso práctico. Imaginemos que producto de un estudio obtenemos n observaciones, cada una con p atributos, de tal forma que cada una se puede clasificar binariamente. Se puede pensar, por ejemplo, en un estudio para determinar si una persona padece alguna enfermedad o no (clasificación binaria) con base en atributos cuantitativos (edad, peso, actividad física, consumo de azúcar, etc.). Para simplificar esta primera aproximación, supongamos que $p = 2$, con lo que las n observaciones del estudio se pueden representar en un plano:

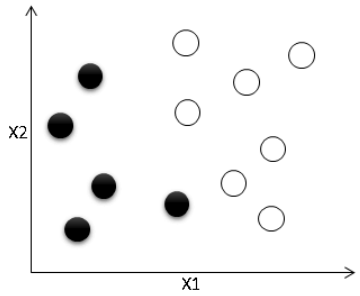


Figura 1: Los ejes representan cada atributo, y el color negro o blanco la clasificación binaria (si padece la enfermedad o no).

Se puede observar que, en este caso en particular, los conjuntos blanco y negro son linealmente separables. Es decir, podemos encontrar un recta que separe uno de otro. Más aún, es fácil ver que son infinitas las rectas que cumplen con esto (Figura 2).

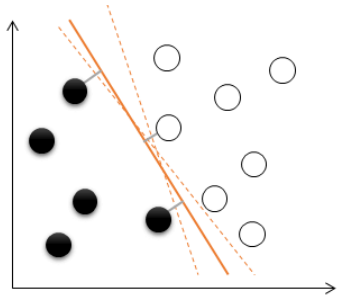


Figura 2:

El método de SVM's nos permitirá encontrar la recta que maximice la distancia de ésta respecto a ambos conjuntos. Para entenderlo, hablemos un poco de las matemáticas detrás de todo esto.

Sea $\beta_0 + \omega_1 x_1 + \omega_2 x_2 = 0$ la ecuación de una recta que divide estos conjuntos. Esto nos dice que, si tomamos un punto arbitrario del plano, $\vec{X'} = x'_1, x'_2$, y sustituimos los valores x'_1 y x'_2 en las ecuación de la recta de arriba, tendremos tres posibilidades:

$$\begin{aligned}
&\beta_0 + \omega_1 x_1 + \omega_2 x_2 = 0, && \text{El punto se encuentra en la recta.} && \\
&\beta_0 + \omega_1 x_1 + \omega_2 x_2 > 0, && \text{El punto se encuentra a la derecha de la recta (bola blanca).} && (1)\\
&\beta_0 + \omega_1 x_1 + \omega_2 x_2 < 0, && \text{El punto se encuentra a la izquierda de la recta (bola negra).} && (2)
\end{aligned}$$

Las ecuaciones (1) y (2) usualmente se conocen como *reglas de decisión*. Ahora, en lugar de bola negra o bola blanca, denotemos las dos clases que tenemos como $y_i = \{-1, +1\}$, reescribamos las ecuaciones anteriores y multipliquémoslas por y_i:

$$\begin{aligned}
&\beta_0 + \vec{w} \cdot \vec{x} > 0 \Leftrightarrow y_i(\beta_0 + \vec{w} \cdot \vec{x}) > 0, && \text{Con } y_i = +1. && (3)\\
&\beta_0 + \vec{w} \cdot \vec{x} < 0 \Leftrightarrow y_i(\beta_0 + \vec{w} \cdot \vec{x}) > 0, && \text{Con } y_i = -1 && (4)
\end{aligned}$$

Es decir, de esta manera encontramos una sola desigualdad para representar las dos clases:

$$y_i(\beta_0 + \vec{w} \cdot \vec{x}) \geq 0, \qquad \text{Con } y_i = \{-1, +1\} \tag{5}$$

Introducimos ahora un nuevo parámetro τ, conocido como el *margen*, que corresponde a la distancia mínima entre la recta y el ejemplo cualquiera más próximo.

Así pues, la recta (o *hiperplano*, cuando $p > 2$) de separación óptima será aquella donde $\tau = min\{\tau_1, \tau_2\}$ sea máximo. Es inmediato ver que la recta equidista a los vectores de soporte cuando es óptica (Figura 4).

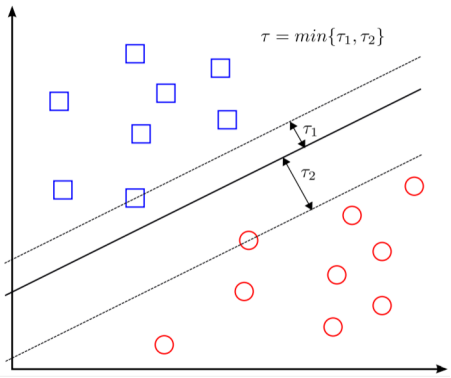


Figura 3: [3] Las figuras (vectores) por donde pasan las líneas punteadas son las más cercanos a la propuesta de recta elegida, por lo que indican los valores de τ_1 y τ_2. El valor τ será el más pequeño de estos dos. A estas figuras (vectores) se les conoce como *vectores de soporte*, porque precisamente son los que constriñen al margen.

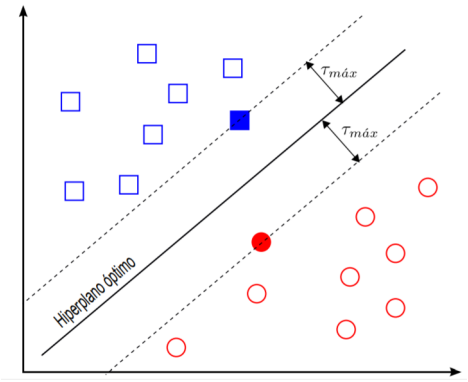


Figura 4: [3]

Introducimos este nuevo parámetro en las ecuaciones (3) y (4), las que nos permiten asignarles valores positivos o negativos a cada uno de los ejemplos:

$$y_i(\beta_0 + \vec{w} \cdot \vec{x}) \geq \tau, \qquad \text{Con } y_i = \{-1, +1\} \tag{6}$$

Donde se cumple la igualdad en las fronteras del «canal» (o «avenida»).

$$y_i(\beta_0 + \vec{w} \cdot \vec{x}) = \tau, \qquad \forall i \in V_f \tag{7}$$

Con V_f el conjunto de todos los vectores en las fronteras del canal.

Con esto en mente, calculemos el ancho del canal. Para lo cual consideremos dos puntos arbitrarios, $\vec{X_+}$ y $\vec{X_-}$, situados justo en las fronteras positiva y negativa, como se muestra en la Figura 5. El vector que los una será $\vec{X_+} - \vec{X_-}$. Además, por construcción de la recta $(\beta_0 + \vec{w} \cdot \vec{x} = 0)$, $\vec{w}$ es un vector normal a ésta.

De tal forma que el ancho del canal está dado por:

$$a = \left(\vec{X_+} - \vec{X_-}\right) \cdot \frac{\vec{w}}{||\vec{w}||}$$

Pero como el producto interior es distributivo:

$$a = \left(\vec{X_+} \cdot \vec{w} - \vec{X_-} \cdot \vec{w}\right) \frac{1}{||\vec{w}||} \tag{8}$$

Retomemos la ecuación (7) y démonos cuenta que, para muestras positivas ($y_i = +1$):

$$\vec{X_+} \cdot \vec{w} = \tau - \beta_0$$

Para $y_i = -1$:

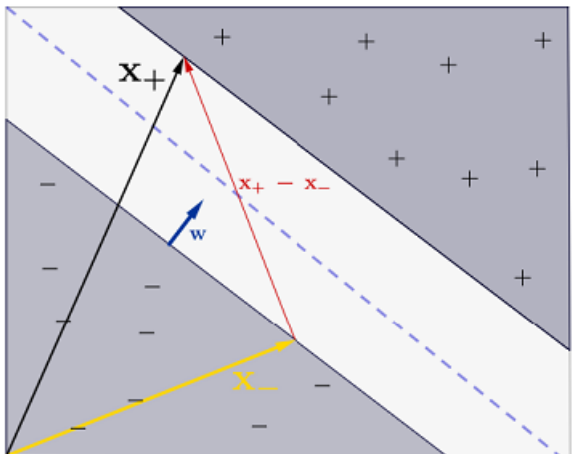
$$\vec{X_-} \cdot \vec{w} = -\tau - \beta_0$$

Por lo que la ecuación (8) resulta:

$$a = \frac{2\tau}{||\vec{w}||} \tag{9}$$

Para limitar el número de soluciones que tiene la ecuación (8), se llegó a la convención [3] de escalar el τ y $||\vec{w}||$ a la unidad. Es decir:

$$\tau||\vec{w}|| = 1$$



De tal manera que, si definimos la recta (o el hiperplano, como se verá más adelante) de separación como la función lineal $D(\vec{\boldsymbol{x}}) = \vec{\boldsymbol{x}} \cdot \vec{\omega} + \beta_0$, se tenga:

$$D(\vec{X_+}) = +1 \tag{10}$$

$$D(\vec{X_-}) = -1 \tag{11}$$

Con lo que la ecuación (9) resulta ser:

$$a = \frac{2}{||\vec{w}||^2}$$

Así, el problema de obtener el hiperplano óptimo se reduce a calcular el máximo valor que pueda tener a, que a su vez se reduce a calcular el mínimo de $||\vec{w}||^2$. O bien, el mínimo de $\frac{1}{2}||\vec{w}||^2$, por ser matemáticamente más conveniente. Podemos plantearlo de la siguiente manera:

Problema primal. Encontrar

$$min \frac{1}{2}||\vec{w}||^2$$

dada la constricción:

$$y_i(\beta_0 + \vec{w} \cdot \vec{x_i}) - 1 \geq 0, \qquad i = 1, 2, 3...n \tag{12}$$

Para esto, se utiliza la técnica de los multiplicadores de Lagrange y el Teorema Karush-Kuhn-Tucker para transformar este problema primal en uno dual, que generalmente es más sencillo de resolver.

En primer lugar, se construye una función lagrangiana de la forma:

$$L(\vec{w}, b, \alpha_i) = f(\vec{\omega}, b) - \sum \alpha_i g_i(\vec{\omega}, b) \tag{13}$$

Donde $g_i(\vec{\omega, b})$ es la constricción del problema. Así, la lagrangiana en nuestro caso resulta ser:

$$L(\vec{w}, b, \alpha_i) = \frac{1}{2}\vec{w}^2 - \sum \alpha_i \left[y_1 \left(\vec{w} \cdot \vec{x}_i + b\right) - 1\right] \tag{14}$$

con $\alpha_i \geq 0$ siendo los multiplicadores de Lagrange. Aplicando las condiciones de Karush-Kuhn-Tucker (ver apéndice 1) se tienen las siguientes proposiciones:

$$\frac{\partial L}{\partial \vec{\omega}} = \vec{\omega} - \sum \alpha_i y_i \vec{x_i} = 0 \Rightarrow \vec{\omega} = \sum_{i=0}^{n} \alpha_i y_i \vec{x_i} \tag{15}$$

$$\frac{\partial L}{\partial b} = -\sum \alpha_i y_i = 0 \tag{16}$$

$$\alpha_i[1 - y_i(\vec{\omega} \cdot \vec{x_i} + \beta)] = 0, \qquad i = 1, 2, ..., n \tag{17}$$

Así, se sustituye (15) en (14):

$$L(\vec{\alpha}) = \frac{1}{2}\left(\sum_{j=0}^{n} \alpha_j y_j x_j\right)\left(\sum_{i=0}^{n} \alpha_i y_i x_i\right) - \left(\sum_{j=0}^{n} \alpha_j y_j x_j\right)\left(\sum_{i=0}^{n} \alpha_i y_i x_i\right) - b\sum_{i=0}^{n} \alpha_i y_i + \sum_{i=0}^{n} \alpha_i$$

Pero, por (12), el tercer sumando se anula y podemos simplificar la expresión anterior como:

$$L(\vec{\alpha}) = -\frac{1}{2}\left(\sum_{j=0}^{n} \alpha_j y_j x_j\right)\left(\sum_{i=0}^{n} \alpha_i y_i x_i\right) + \sum_{i=0}^{n} \alpha_i = \sum_{i=0}^{n} \alpha_i - \frac{1}{2}\sum_{i,j=0}^{n} \alpha_i \alpha_j y_i y_j x_i x_j$$

Llegando así al *Problema dual:*

$$max \quad L(\vec{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i,j=0}^{n} \alpha_i \alpha_j y_i y_j \vec{x_i} \cdot \vec{x_j} \tag{18}$$

Con las restricciones:

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

De esta manera, el problema se transforma en encontrar una $\vec{\alpha}$ que maximice la lagrangiana utilizando técnicas de análisis numérico. La razón por la que se obtuvo el dual y no se utilizan directamente técnicas de análisis numérico en (12) es porque el coste computacional de éste último asciende con la dimensionalidad de las muestras, mientras que en (18) lo hace con el número de muestras [3] . Después, sólo basta sustituirla en (11) para calcular $\vec{\omega}$ y, finalmente, reemplazar este valor en la ecuación del hiperplano:

$$D(\vec{\boldsymbol{x}}) = \vec{\omega} \cdot \vec{\mathbf{x}} + \beta_0 = \beta_0 + \omega_1 x_1 + \omega_2 x_2 + ... + \omega_p x_p \tag{19}$$

Donde $\vec{\boldsymbol{x}}$ denota la dimensión del espacio donde graficamos las muestras. Hemos estado trabajando sólo considerando dos atributos medibles a cada una de nuestras observaciones, pero salta a la vista rápidamente que, si tenemos p atributos para cada una de nuestros ejemplos, necesitaremos un espacio $p - dimensional$ donde colocarlos. Así, el hiperplano que los divida será $p - 1$ dimensional y se podrá expresar también como:

$$D(\vec{\boldsymbol{x}}) = \sum_{i=1}^{n} \alpha_i^* y_i \vec{x} \cdot \vec{x_i} + b_0 \tag{20}$$

Es decir, podemos reemplazar cualquier ejemplo en (20), y, dependiendo si obtenemos un valor negativo o positivo, sabremos a cuál clase pertenece. Es por esto que a la ecuación (20) la conoceremos como la **función de decisión.**
Bueno, aún no. Aún no hemos calculado β_0. Esto es fácil utilizando la tercera restricción obtenida del teorema KKT (ecuación 17), donde observamos que, para los casos donde $\alpha_i > 0$, necesariamente:

$$1 - y_i(\vec{\omega} \cdot \vec{x_i} + \beta_0) = 0$$

$$\Rightarrow \beta_0 = y_i - \vec{\omega} \cdot \vec{x_i} \tag{21}$$

Más aún, de las ecuaciones (10) y (11) vemos que *únicamente* los vectores de soporte cumplen que $y_i(\vec{\omega} \cdot \vec{x_i} + \beta_0) = 1$. Con las demás muestras, $y_i(\vec{\omega} \cdot \vec{x_i} + \beta_0) > 1$, por lo que necesariamente sus α_i asociados son nulos, para que se satisfaga la restricción (17). Con esto concluimos dos cosas. 1) la primera, que la ecuación (21) se puede reescribir como:

$$\beta_0 = y_{vs} - \vec{\omega} \cdot \vec{x_{vs}} \tag{22}$$

con $(\vec{x_{vs}}, y_{vs})$ representando la tupla de cualquier vector de soporte.
Y, 2) el hiperplano de separación (20) es sólo una combinación lineal de los vectores de soporte, dado que los demás vectores tendrán asociado un coeficiente $\alpha_i = 0$.

0.1.2. Máquinas de Soporte Vectorial

En la sección anterior trabajamos con un set de datos que pueden ser separados por un hiperplano, siendo ese proceso de clasificación a lo que se conoce como *Clasificador de Soporte Vectorial*. Sin embargo, en la mayoría de los casos los datos no son linearmente separables (Figura 6).

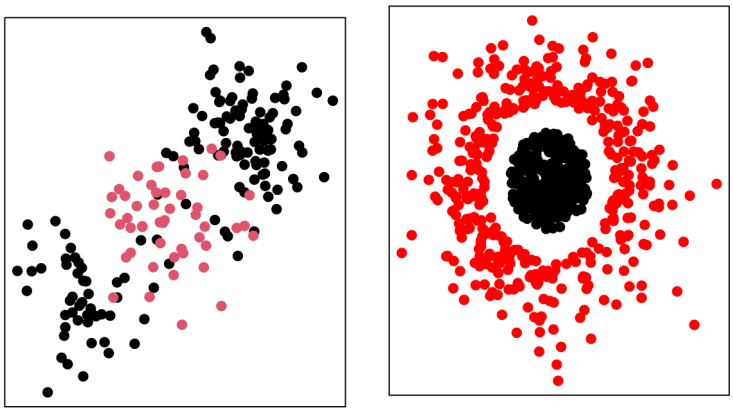


Figura 6: Sets de datos no linearmente separables.

En estos casos, lo que se busca es realizar una transformación del espacio de las observaciones (espacio de entradas) hacia un espacio de mayor dimensión donde sí podamos encontrar un hiperplano de separación (espacio de caracterísiticas). Sea $\Phi : \boldsymbol{X} \rightarrow \boldsymbol{F}$ una transformación que toma un vector $\vec{x}$ y lo lleva a un espacio de características $\boldsymbol{F}$ cuyas bases son $\phi(\vec{x}) = \{\phi_0(\vec{x}), \phi_1(\vec{x}), ..., \phi_n(\vec{x})\}$, donde al menos alguna de éstas es no lineal.

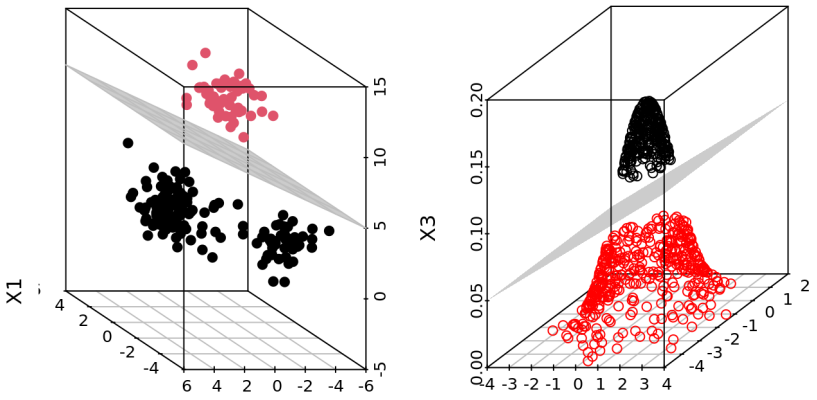


Figura 7: Sets de datos linearmente separables.

Se formula una función de decisión para el nuevo espacio de características, que, de hecho, es equivalente a la expresada en (19):

$$D(\vec{\boldsymbol{x}}) = \vec{\omega} \cdot \vec{\phi}(\vec{\boldsymbol{x}}) \tag{23}$$

Salta a la vista la ausencia del término β_0, que se compensa al añadir la función constante $\phi_0(\vec{\boldsymbol{x}}) = 1$ a la base de funciones de transformación, así como aumentar una dimensión al vector $\vec{\omega}$.
Similarmente a como se hizo en la sección anterior, se encuentra la función de decisión del problema dual:

$$D(\vec{\boldsymbol{x}}) = \sum_{i=0}^{n} \alpha_1^* y_i \vec{\phi}(\vec{\boldsymbol{x}}) \cdot \vec{\phi}(\vec{x_i}) \tag{24}$$

Vale la pena racalcar que lo que se busca es que la función ϕ siempre aumente la dimensión de las muestras (es decir, aumentar el número de funciones bases), en aras de lograr que éstas sean linealmente separables en el espacio de características. Por lo que el espacio de caracteristicas puede llegar a tener una dimension muy grande, incluso a dmension infinito. En la siguiente subsección. en específico, se introducen las funciones kernel, que son de ayuda para simplificar el cálculo.

Función Kernel

La idea de una función de kernel es analizar con mayor facilidad datos en un espacio de características de dimensión grande

Una función kernel es una función $\kappa : \mathbf{X} \times \mathbf{X} \rightarrow \mathbb{R}$ que asigna a cada par de vectores del conjunto de entrada el valor real del producto interior de las imágenes de dichos vectores al aplicarles una transformación lineal $\Phi : \mathbf{X} \rightarrow \mathbf{F}$. Esto es:

$$K(\vec{x_i}, \vec{x_j}) = \vec{\phi}(\vec{x_i})^T \vec{\phi}(\vec{x_j}) = \vec{\phi}(\vec{x_i}) \cdot \vec{\phi}(\vec{x_j}) \tag{25}$$

Se reescribe la regla de decisión (24):

$$D(\vec{\boldsymbol{x}}) = \sum_{i=0}^{n} \alpha_1^* y_i K(\vec{\boldsymbol{x}}, \vec{x_i}) \tag{26}$$

Así, la ecuación (26) es la solución al problema de clasificar ejemplos no separables linealmente, donde los coeficientes α_1^* son aquellos que maximizan la siguiente lagrangiana (problema de optimización):

$$max \quad L(\vec{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=0}^{n} \alpha_i \alpha_j y_i y_j K(\vec{x_i}, \vec{x_j}) \tag{27}$$

Con las restricciones:

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

Donde se conoce el conjunto de datos de entrenamiento $(\vec{x_i}, y_i)$ y la función kernel.

A partir de la función kernel se logra no hacer operaciones explicitas en el espacio de características, sino que trabaja en el domnio del espacio de la datos original

Ejemplos de funciones Kernel

Una forma de definir las funciones kernel es [6]

Sea X un conjunto no vacio. Una función $\kappa : \mathbf{X} \times \mathbf{X} \to \mathbb{R}$ es una función kernel si la matriz K con entradas $K_{m,m'} = \kappa\left(x^m, x^{m'}\right)$ es semipositiva.Quiere decir

$$\sum_{m,m'=1}^{M} c_m c_{m'}^* \kappa\left(x^m, x^{m'}\right) \geq 0 \tag{28}$$

Esto nos asegura cualquier función con una imagen positiva puede ser considerada una función kernel

Algunos ejemplos comunes de funciones Kernel son:

1. Kernel lineal: $K_L(\vec{x_i}, \vec{x_j}) = \vec{x_i} \cdot \vec{x_j}$
2. Kernel polinómico de grado-p: $K_P(\vec{x_i}, \vec{x_j}) = [\gamma(\vec{x_i} \cdot \vec{x_j}) + \tau]^p$ con $\gamma > 0$
3. Kernel sigmoidal: $K_s(\vec{x_i}, \vec{x_j}) = tanh[\gamma(\vec{x_i} \cdot \vec{x_j}) + \tau]$ con $\gamma > 0$
4. Kernel de base radial (RBF): $K_r(\vec{x_i}, \vec{x_j}) = exp\left(-\frac{||\vec{x_i} - \vec{x_j}||}{2\sigma^2}\right)$

0.2. Máquinas de Soporte Vectorial Cuánticas

En esta sección discutiremos las dos principales aproximaciones a una máquina de soporte vectorial cuántica: el clasificador cuántico variacional y el estimador de kernel cuántico. Ambos parten de una idea muy poderosa: codificar la base de datos en un estado cuántico en el espacio de Hilbert, de tal manera que éste cumpla el papel de mapa de características cuántico.
Como se verá más adelante, la diferencia entre estos dos métodos radica en que en el primero se usa un circuito variacional cuántico para recrear una SVM, y en el segundo sólo se utiliza una computadora cuántica para calcular la matriz kernel que será utilizada para procesar los datos clásicamente, como se explicó en la sección 1.

0.2.1. Espacios de Características Cuánticos

La manera en la que se pueden codificar los datos clásicos a una forma cuántica es definiendo una compuerta $U_{\phi(\vec{x})}$ que se le aplica al estado inicial $|0\rangle^{\otimes n}$ (elegido por convención). Así, la representación de un vector $\vec{x}$ en un circuito cuántico será $U_{\phi(\vec{x})} |0\rangle^{\otimes n}$, donde n es el número de qubits que se estén utilizando. Analicemos algunos ejemplos de mapas de características, usando las compuertas genéricas U_1 y U_3.

La compuerta genérica U_1 necesita de un sólo ángulo para implementar una rotación sobre un estado inicial $|0\rangle$. Si suponemos que nuestro conjunto de datos iniciales tiene tres dimensiones, $\vec{x} = (x_1, x_2, x_3)$, se puede definir una función lineal $\varphi : x_i \in \mathbf{R} \to (0, 2\pi]$, para introducir los datos de entrada en el circuito como $|\phi(\vec{x})\rangle = U_1(\varphi(\vec{x})) |0\rangle^{\otimes n}$. Es decir, en el caso de tres qubits:

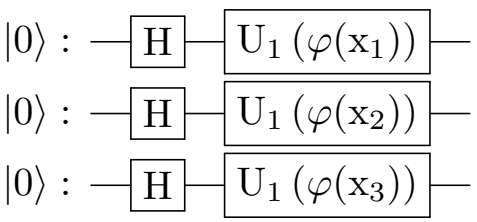
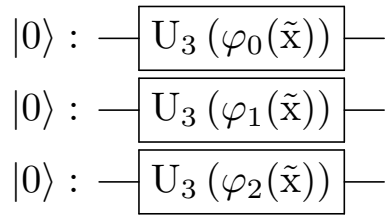


Figura 8: Caption

Con $U_1(\lambda) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\lambda} \end{pmatrix}$.

Para la compuerta U_3 se necesitan tres ángulos, por lo cual ahora se debe utilizar una función no lineal $\varphi : \vec{x} \to (0, 2\pi] \times (0, 2\pi] \times (0, \pi]$:



Con $U_3(\theta, \alpha, \lambda) = \begin{pmatrix} cos\frac{\theta}{2} & -e^{i\lambda}sin\frac{\theta}{2} \\ e^{i\alpha}sin\frac{\theta}{2} & e^{i(\alpha+\lambda)}cos\frac{\theta}{2} \end{pmatrix}$.

Aunque estos ejemplos son los más intuitivos e ilustrativos, para obtener una ventaja sobre enfoques clásicos se necesita implementar un mapeo basado en circuitos cuánticos que son difíciles de simular clásicamente. Se ha demostrado [7] que uno de estos mapeos es el descrito por la siguiente expresión:

$$\mathcal{U}_\Phi(\vec{x}) = U_{\Phi(\vec{x})} H^{\otimes n} U_{\Phi(\vec{x})} H^{\otimes n} \tag{29}$$

donde

$$U_{\Phi(\vec{x})} = exp\left(i \sum_{S \subseteq [n]} \varphi_s(\vec{x}) \prod_{i \in S} Z_i \right) \tag{30}$$

La función φ_S que se usa por default en este caso es:

$$\varphi_S : \vec{x} \to \begin{cases} x_i & si \quad S = \{i\} \\ (\pi - x_i)(\pi - x_j) & si \quad S = \{i, j\} \end{cases} \tag{31}$$

Además, los vectores de entrada $\vec{x}$ pasan por un preprocesamiento para normalizarlos y escalarlos en el intervalo $[-1, 1]$. Este mapeo en particular nos será de ayuda en la parte final de este manual, y en las siguientes secciones analizaremos cómo podemos manipular estos vectores de características.

0.2.2. Clasificador Cuántico Variacional

Este método hace uso de lo que se conoce como como **circuito cuántico variacional** para realizar tres tareas:

- Codificar los datos en un estado cuántico en el circuito.
- Asignarle una etiqueta (-1,+1) a cada salida.
- Optimizar los parámetros implicados en el circuito para mejorar los resultados.

La primera tarea se resuelve aplicando una compuerta $U_{\phi_{\vec{x}}}$ a los estados $|0\rangle^{\otimes n}$. Luego, para de obtener algún valor de un circuito cuántico, hay que medir. En este ejercicio, nos conviene hacerlo con la observable Z, pues sabemos que sus eigenvalores son -1 y +1:

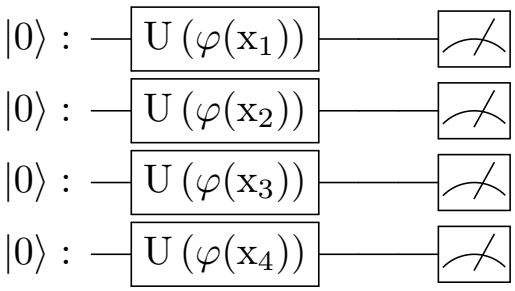


Figura 9: Caption

Ahora, podemos agregar algunas compuertas intermedias que aprovechen el entrelazamiento y, además, varíen en función de un parámetro $\vec{\theta} = (\theta_1, \theta_2, \theta_3...)$.

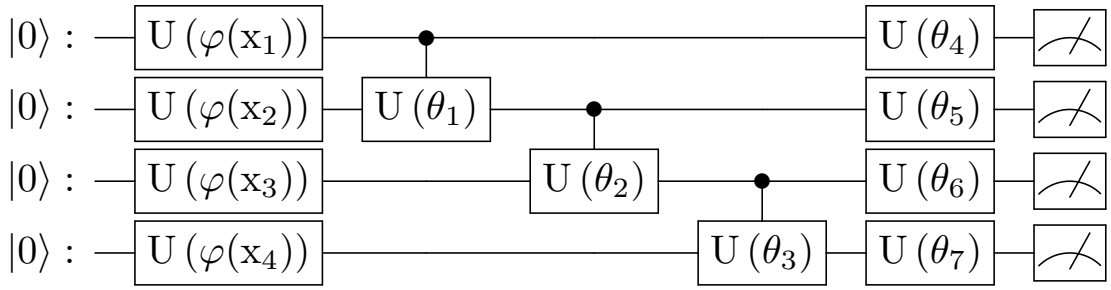


Figura 10: Caption

Podemos compactar este circuito como se muestra en Figura 11 :

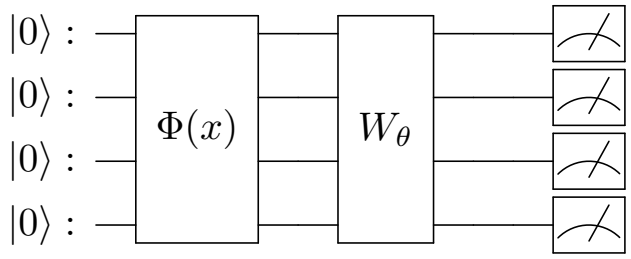


Figura 11: **Circuito Cuántico Variacional**: $\Phi(\vec{x})$ representa los datos de entrada; es el mapa de características cuántico que resulta ser no otra cosa que un subcircuito del CCV. $W_{\vec{\theta}}$ cumple el papel de ser un optimizador de la observable que vamos a medir.

En este punto, ya somos capaces de escribir una regla de decisión como las que tenemos en el primer capítulo. Teniendo una $\vec{\theta}$ óptima, podemos saber a cuál clase pertenece un elemento $\vec{x}$, evaluando la función:

$$f_{\vec{\theta}}(\vec{x}) = \langle \phi(\vec{x}) | \, W_{\vec{\theta}}^{\dagger} Z W_{\vec{\theta}} | \phi(\vec{x}) \rangle \in [-1, 1] \tag{32}$$

Se elige un umbral $b \in [-1, 1]$, de tal manera que:

$$clase(\vec{x}) = \begin{cases} +1 & si \quad f_{\vec{\theta}}(\vec{x}) \geq b \\ -1 & si \quad f_{\vec{\theta}}(\vec{x}) < b \end{cases} \tag{33}$$

Recapitulando un poco, hemos encontrado un *Clasificador Variacional Cuántico* (CVC), ¿pero éste qué tiene que ver con las Máquinas de Soporte Vectorial, que son clasificadores lineales? El caso es que se puede demostrar que los CVC son también clasificadores lineales. A continuación escribiremos la prueba.

Definamos una nueva observable $H_{\vec{\theta}} = W_{\vec{\theta}}^{\dagger} Z W_{\vec{\theta}}$ y reescribamos la regla de decisión (32):

$$f_{\vec{\theta}}(\vec{x}) = \langle \phi(\vec{x}) | \, H_{\vec{\theta}} | \phi(\vec{x}) \rangle \tag{34}$$

Este resultado es un escalar y sabemos que la $Tr[c] = c$, con c un escalar. Entonces:

$$\langle \phi(\vec{x}) | \, H_{\vec{\theta}} | \phi(\vec{x}) \rangle = Tr[\langle \phi(\vec{x}) | \, H_{\vec{\theta}} | \phi(\vec{x}) \rangle] \tag{35}$$

Lo que nos permite aplicar la propiedad $Tr[AB] = Tr[BA]$.

$$Tr[\langle \phi(\vec{x}) | \, H_{\vec{\theta}} | \phi(\vec{x}) \rangle] = Tr[H_{\vec{\theta}} | \phi(\vec{x}) \rangle \langle \phi(\vec{x}) |] \tag{36}$$

Podemos reconocer fácilmente a la matriz de densidad, que es otra forma de almacenar la información de los vectores de estado. Consecuentemente, definamos $\Phi(\vec{x}) = |\phi(\vec{x})\rangle \langle\phi(\vec{x})|$, que vive en el espacio $\mathcal{C}^{2^n,2^n}$, y reescribamos (36):

$$f_\theta(\vec{x}) = Tr[H_{\vec{\theta}}\Phi(\vec{x})] \tag{37}$$

Sabemos que una observable se puede descomponer en una suma de matrices de Pauli. Más aún, una matriz de densidad, al ser hermitiana, también puede sufrir una descomposición así:

$$H_\theta = \frac{1}{2^n}\sum_{\alpha\in 4^n}\langle H_\theta, P_\alpha\rangle_{HS} P_\alpha, \qquad \phi(\vec{x}) = \frac{1}{2^n}\sum_{\alpha\in 4^n}\langle \Phi(\vec{x}), P_\alpha\rangle_{HS} P_\alpha \tag{38}$$

Donde P_α son las matrices de Pauli (incluyendo la identidad) y el producto interno $\langle,\rangle_{HS}$ es el producto interno Hilbert-Schmit. Éste se define: $\langle A, B\rangle_{HS} = Tr[A^*B]$. Las ecuaciones (38) resultan:

$$H_\theta = \frac{1}{2^n}\sum_{\alpha\in 4^n} h_\alpha P_\alpha, \qquad \phi(\vec{x}) = \frac{1}{2^n}\sum_{\alpha\in 4^n} \Phi_\alpha(\vec{x}) P_\alpha \tag{39}$$

Con $h_\alpha = Tr[H_\theta P_\alpha]$ y $\phi_\alpha(\vec{x}) = Tr[\Phi(\vec{x})P_\alpha]$.

Así que, reemplazando estas expresiones en (37):

$$f_\theta(\vec{x}) = Tr\left[\left(\frac{1}{2^n}\sum_{\alpha\in 4^n} h_\alpha P_\alpha\right)\left(\frac{1}{2^n}\sum_{\beta\in 4^n}\Phi_\beta(\vec{x})P_\beta\right)\right] \tag{40}$$

$$= \frac{1}{4^n} Tr\left[\left(\sum_{\alpha\in 4^n} h_\alpha P_\alpha\right)\left(\sum_{\beta\in 4^n}\Phi_\beta(\vec{x})P_\beta\right)\right] = \frac{1}{4^n} Tr\left[\sum_{\alpha,\beta\in 4^n} h_\alpha \Phi_\beta(\vec{x}) P_\alpha P_\beta\right] \tag{41}$$

Como la traza es una operación lineal, la podemos distribuir en la suma:

$$f_\theta(\vec{x}) = \frac{1}{4^n}\sum_{\alpha,\beta\in 4^n} h_\alpha \Phi_\beta(\vec{x}) Tr[P_\alpha P_\beta] \tag{42}$$

Expresar nuestra de regla de decisión de esta manera tiene la ventaja de que, como las matrices de Pauli son ortogonales entre sí bajo el producto Hilbert-Schmit. Es decir:

$$Tr[P_\alpha P_\beta] = \begin{cases} 2^n & si \quad \alpha = \beta \\ 0 & si \quad \alpha \neq \beta \end{cases} \tag{43}$$

Lo que implica que:

$$f_\theta(\vec{x}) = \frac{1}{2^n}\sum_{\alpha\in 4^n} h_\alpha \Phi_\alpha(\vec{x}) \tag{44}$$

Si agregamos nuevamente el parámetro umbral b, dándonos cuenta que realmente (44) sigue estando acotada en $[-1, 1]$, podemos darnos cuenta que la función de decisión resultante es:

$$clase(\vec{x}) = signo\left(\frac{1}{2^n}\sum_{\alpha\in 4^n} h_\alpha \Phi_\alpha(\vec{x}) + b\right) \tag{45}$$

En esta última expresión resulta más obvio que se trata de un clasificador lineal, como los descritos en las ecuaciones (19) y (23). Más aún, nos provee de información acerca de qué es físicamente la observable $W_{\vec{\theta}}$: una parametrización de un pequeño subconjunto de posibles hiperplanos de separación.

0.2.3. Estimador de Kernel Cuántico

Es otro método para implementar una computadora cuántica la clasificación de datos tambien planteado por Havlíček et al. [7] es utilizar un algoritmo cuántico para estimar el kernel de los pares de datos y utilizar los resultados en el problema clásico (0.1.2).

Para eso utilizaremos el espacio de Hilbert como espacio de caracteristicas cuántico introduciendo nuestros datos a un circuito unitario como se vio en la sección anterior $U_\phi(x)\left|0^{\otimes n}\right\rangle = \left|\phi\left(\mathbf{x}\right)\right\rangle\left\langle\phi\left(\mathbf{x}\right)\right|$, y si consideramos la definición de la matriz de kernel como el producto interior descrito en la sección clásica (25) podemos pensar en las entradas de la matriz de kernel cuántica como $K_{ij} = \left|\left\langle\phi^\dagger\left(\mathbf{x}_j\right) \mid \phi\left(\mathbf{x}_i\right)\right\rangle\right|^2$, y por lo tanto encontrar sus entradas a partir de las amplitudes de transición

$$K_{ij} = \left|\left\langle\phi^\dagger\left(\mathbf{x}_j\right) \mid \phi\left(\mathbf{x}_i\right)\right\rangle\right|^2 = \left|\left\langle 0^{\otimes n}\left|\mathbf{U}_\phi^\dagger\left(\mathbf{x}_j\right)\mathbf{U}_\phi\left(\mathbf{x_i}\right)\right|0^{\otimes n}\right\rangle\right|^2 \tag{46}$$

Las cuáles son faciles de conseguir a partir de mediciones repetidas El circuito necesario consiste en:

1. Preparamos los qubits en el estado $|0\rangle^{\otimes n}$
2. Aplicamos el operador unitario $U(x_i)$
3. Aplicamos el operador adjunto $U^\dagger(x_j)$
4. Medimos en la base Z

Repetimos en proceso un número R de veces y la frecuencia con la obtengamos una cadena de ceros será el valor estimado de la entrada k_{ij} de la matriz de kernel junto con un error aditivo ϵ.

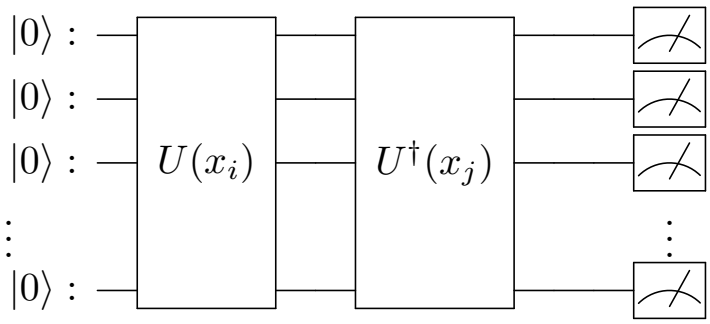


Figura 12:

0.3. Programando la Máquina de Soporte Vectorial Cuántica

Como mencionamos anteriormente, la eficacia de las máquinas de soporte vectorial radica en el hecho de encontrar el hiperplano de separación entre dos conjuntos originalmente no linealmente separables, utilizando funciones kernel. En esta sección, desarrollaremos un ejercicio de aprendizaje supervisado utilizando la versión clásica y la cuántica de este algoritmo, con el objetivo de determinar si existe alguna ventaja cuántica.

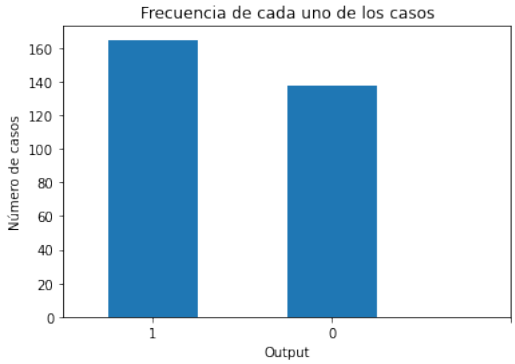
0.3.1. Set de datos

Como ejemplo utilizaremos un dataset de dominio público llamado "Heart Disease Data Set"[9] con 303 entradas y 14 columnas en cada una. Estos datos pretenden ser de utilidad para el diagnóstico de afecciones en el arteria coronaria.

	age	sex	cp	trtbps	chol	fbs	restecg	thalachh	exng	oldpeak	slp	caa	thall	output
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
298	57	0	0	140	241	0	1	123	1	0.2	1	0	3	0
299	45	1	3	110	264	0	1	132	0	1.2	1	0	3	0
300	68	1	0	144	193	1	1	141	0	3.4	1	2	3	0
301	57	1	0	130	131	0	1	115	1	1.2	1	1	3	0
302	57	0	1	130	236	0	0	174	0	0.0	1	1	2	0

Las variables corresponden a la edad del paciente, su sexo, el tipo de dolor que expresa sentir en el pecho (anginal regular, angina atípica, no en la angina o asintomático), la presión sanguínea en mm de Hg, el porcentaje de colesterol sérico, si la glucemia en ayunas es mayor a 120mg/dl, los resultados de su electrocardiograma, el máximo ritmo cardíaco alcanzado, si hubo angina inducida por el ejercicio, etc. Para más información sobre el dataset, se recomienda visitar la referencia (9). Así, en este set de datos, cada input se puede representar con un vector de 13 entradas. El objetivo de este problema es el de construir un clasificador que

La frecuencia de cada uno de los casos (0 si no hay presencia de afección, 1 si sí) se puede ver en el siguiente histograma.



0.3.2. Máquina de Soporte Vectorial Clásica

Para la implementación clásica utilizaremos las funciones de la biblioteca Sklearn y los kernels integrados más comunes.

Separamos el vector de etiquetas de las demás columnas, así como dividimos el set de datos en 80 % y 20 % para el conjunto de entrenamiento y prueba, respectivamente.

Las bibliotecas que necesitamos importar son las siguientes:

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

```
#Librerias de machine learning clásico
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler, MinMaxScaler
```

```
#Librerias de computo cuántico
from qiskit import Aer, execute
from qiskit.circuit import QuantumCircuit, Parameter, ParameterVector
from qiskit.circuit.library import PauliFeatureMap, ZFeatureMap, ZZFeatureMap
from qiskit.circuit.library import TwoLocal, NLocal, RealAmplitudes, EfficientSU2
from qiskit.circuit.library import HGate, RXGate, RYGate, RZGate, CXGate, CRXGate, CRZGate
from qiskit_machine_learning.kernels import QuantumKernel
```

A continuación creamos dos arreglos, el primero donde se encuentran los vectores con 13 atributos, y uno más, en forma de columna, con las etiquetas que le corresponden a cada uno de esos vectores (0 ó 1).

```
X = np.array(dataframe.drop(['output'],1))
Output = np.array(dataframe['output'])
```

```
entrenamiento, prueba, entrenamiento_tag, prueba_tag = train_test_split(
     X, Output, test_size=0.2, random_state=22)
print('Datos del entrenamiento: ',entrenamiento.shape)
print('Datos de la prueba: ',prueba.shape)
```

```
Datos del entrenamiento:  (242, 13)
Datos de la prueba:  (61, 13)
```

En las siguientes líneas, escalamos, normalizamos y cortamos nuestros datos.

```
escala = StandardScaler().fit(entrenamiento)
entrenamiento = escala.transform(entrenamiento)
prueba = escala.transform(prueba)

samples = np.append(entrenamiento, prueba, axis=0)
minmax_scale = MinMaxScaler((-1, 1)).fit(samples)
entrenamiento = minmax_scale.transform(entrenamiento)
prueba = minmax_scale.transform(prueba)
```

```
entrenamiento_size = 250
entrenamiento = entrenamiento[:entrenamiento_size]
entrenamiento_tag= entrenamiento_tag[:entrenamiento_size]

prueba_size = 50
prueba= prueba[:prueba_size]
prueba_tag = prueba_tag[:prueba_size]
```

A continuación, implementamos la función de Máquinas de Soporte Vectorial incluida en Sklearn, utilizando los kernels lineal, polinomial, rbf y sigmoide mencionados en la sección anterior.

In [59]:
```
kernels_clasicos = ['linear', 'poly', 'rbf', 'sigmoid']

for kernel in kernels_clasicos:
    svc_clasico = SVC(kernel=kernel)
    svc_clasico.fit(entrenamiento, entrenamiento_tag)
    puntuacion = svc_clasico.score(prueba, prueba_tag)

    sol =svc_clasico.predict([ejemplo])

    print('%s kernel classification test score:  %0.2f' % (kernel, puntuacion))
    print('Predicción: ', sol)
```

```
linear kernel classification test score:  0.80
Predicción:  [1]
poly kernel classification test score:  0.90
Predicción:  [1]
rbf kernel classification test score:  0.85
Predicción:  [1]
sigmoid kernel classification test score:  0.75
Predicción:  [0]
```

0.3.3. Resolución del problema con un kernel cuántico (Máquina de Soporte Vectorial Cuántica)

Qiskitaaaa

Como se mencionó anteriormente, una máquina de soporte vectorial cuántica funciona de la misma forma que una clásica, con la única diferencia de que se utiliza una compuerta cuántica en un circuito a manera de transformación del espacio de entradas a un espacio de características cuántico. A saber:

$$\left|\langle\phi^\dagger(\mathbf{x}_j)|\phi(\mathbf{x}_i)\rangle\right|^2 = \left|\langle 0^{\otimes n}|\mathbf{U}_\phi^\dagger(\mathbf{x}_j)\mathbf{U}_\phi(\mathbf{x_i})|0^{\otimes n}\rangle\right|^2$$

Así,

$$\left|\langle 0^{\otimes n}|\mathbf{U}_\phi^\dagger(\mathbf{x}_j)\mathbf{U}_\phi(\mathbf{x_i})|0^{\otimes n}\rangle\right|^2$$

es el valor que tendrá cada uno de los productos escalares en el problema de maximización de

$$L(\vec{\alpha}) = \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j=0}^{n}\alpha_i\alpha_j y_i y_j K(\vec{x_i},\vec{x_j})$$

.

Hay varias formas en las que se puede obtener los valores anteriores. En el que nos enfocaremos es en el de crear el circuito

$$\mathbf{U}_\phi^\dagger(\mathbf{x}_j)\mathbf{U}_\phi(\mathbf{x_i})$$

y medir la probabilidad de obtener

$$|0^{\otimes n}\rangle$$

.

La compuertas más utilizadas para este propósito son las conocidas como Pauli Feature Maps, que se pueden describir como sigue:

$$\mathcal{U}_{\Phi(\mathbf{x})} = \prod_d U_{\Phi(\mathbf{x})}H^{\otimes n},\ U_{\Phi(\mathbf{x})} = \exp\left(i\sum_{S\subseteq[n]}\phi_S(\mathbf{x})\prod_{k\in S}P_i\right).$$

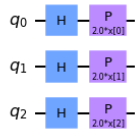
Donde $P_i \in \{I, X, Y, Z\}$, y S es un par ordenado que indica las conexiones que se necesitan para calcular ϕ_S : $S \in \{\binom{n}{k}\ combinaciones,\ k = 1, ...n\}$. Aunque phi_S puede variar, la usada por default es:

$$\phi_S : \mathbf{x} \mapsto \begin{cases} x_i & \text{if } S = \{i\} \\ (\pi - x_i)(\pi - x_j) & \text{if } S = \{i, j\} \end{cases}$$

Por ejemplo $k = 1, P_0 = Z$, obtenemos:

$$\mathcal{U}_{\Phi(\mathbf{x})} = \left(\exp\left(i \sum_j \phi_{\{j\}}(\mathbf{x})\, Z_j \right) H^{\otimes n} \right)^d .$$

Que es conocida como 'ZFeatureMap' y biene incluida en Qiskit.



Teniendo en cuenta que, si X es una matriz,

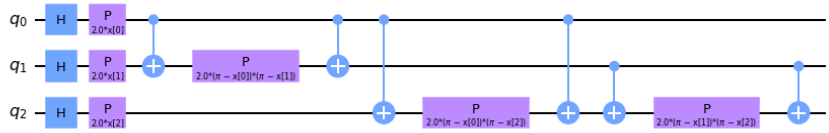
$$e^X = \sum_{k=0}^{\text{ínf}} \frac{X^k}{k!}$$

, podemos desarrollar la parte exponencial de la ecuación del mapa de características de Pauli y llegar a la compuerta fase (P) a un ángulo de 2ϕ, cuando k=1.

Un ejemplo más interesante es el mapa de características que obtenemos al hacer and $k = 2, P_0 = Z, P_1 = ZZ$:

$$\mathcal{U}_{\Phi(\mathbf{x})} = \left(\exp\left(i \sum_{jk} \phi_{\{j,k\}}(\mathbf{x})\, Z_j \otimes Z_k \right) \exp\left(i \sum_j \phi_{\{j\}}(\mathbf{x})\, Z_j \right) H^{\otimes n} \right)^d .$$

Que está integrada en Qiskit como la compuerta 'ZZFeatureMap' y se puede visualizar así:



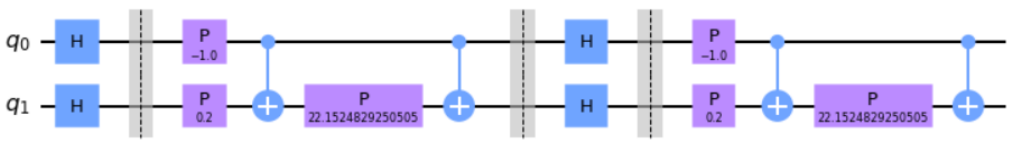
Un ejemplo sencillo de cómo podemos codificar información usando 'ZZFeatureMap´, codifiquemos el vector $x = [-0{,}5, 0{,}1]$:

Un ejemplo sencillo de cómo podemos codificar información usando `ZZFeatureMap`:

In [62]:
```
x = [-0.5,0.1]

xcod = ZZFeatureMap(feature_dimension=2, reps=2, entanglement='linear', insert_barriers=True)
circuito = xcod.bind_parameters(x)
circuito.decompose().draw(output='mpl')
```

Out[62]:



Ahora, un circuito $\mathbf{U}_\phi^\dagger(\mathbf{x}_j)\mathbf{U}_\phi(\mathbf{x_i})$ para dos vectores ejemplo usando 'ZZFeatureMap', con los vectores $x = [-0{,}5, 0{,}1, 0{,}2]$ y $y = [0{,}2, 0{,}5, 0{,}3]$ sería:

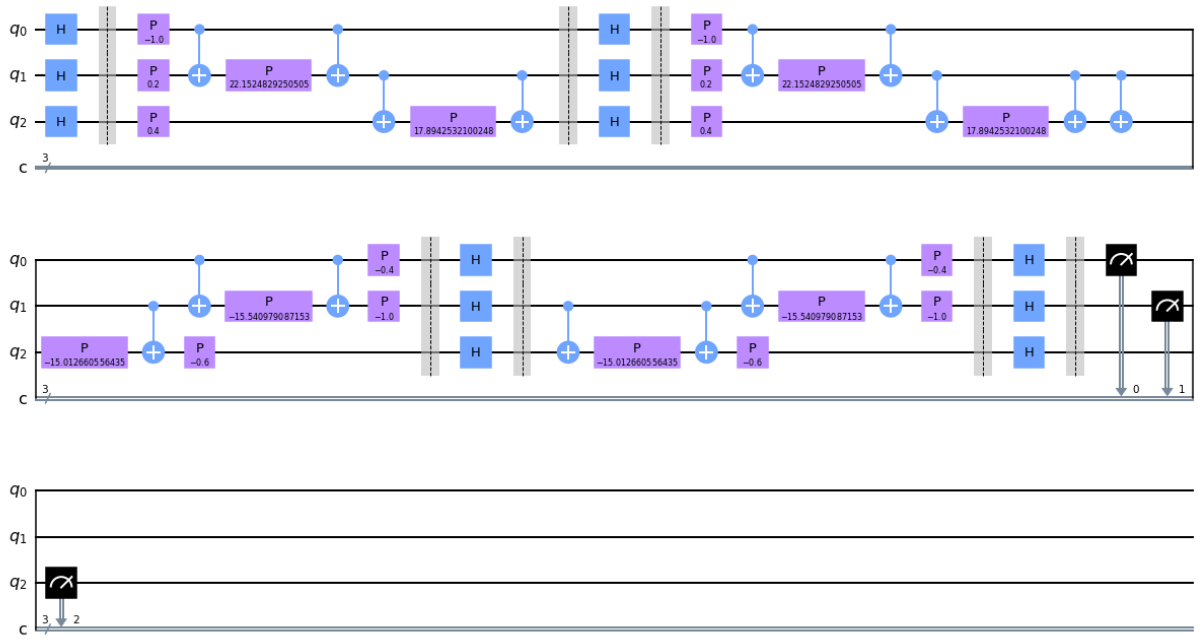
```
In [63]: x1 = [-0.5,0.1,.2]
         y1 = [0.2,0.5,.3]

         zz_map = ZZFeatureMap(feature_dimension=3, reps=2, entanglement='circular', insert_barriers=False)
         zz_kernel = QuantumKernel(feature_map=zz_map, quantum_instance=Aer.get_backend('statevector_simulator'))

         xycod = zz_kernel.construct_circuit(x1, y1)
         xycod.decompose().decompose().draw(output='mpl')
```

Out[63]:

En estas líneas de código utilizamos el Aer Simulator, que es el principal simulador del Aer provider de Qiskit para imitar el funcionamiento de circuitos cuánticos ideales (libres de ruido) en un computador cuántico.[11]



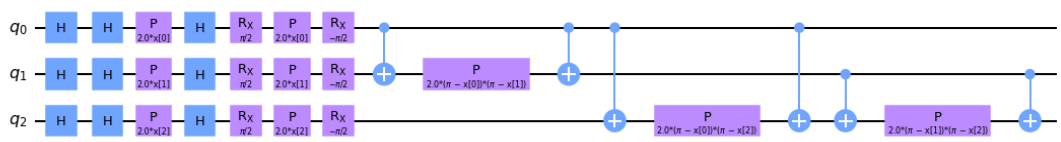
También podemos jugar con las compuertas de esta transformación. Por ejemplo, si hacemos que $P_0 = X, P_1 = Y, P_2 = ZZ$:

$$\mathcal{U}_{\Phi(\mathbf{x})} = \left(\exp\left(i \sum_{jk} \phi_{\{j,k\}}(\mathbf{x})\, Z_j \otimes Z_k \right) \exp\left(i \sum_{j} \phi_{\{j\}}(\mathbf{x})\, Y_j \right) \exp\left(i \sum_{j} \phi_{\{j\}}(\mathbf{x})\, X_j \right) H^{\otimes n} \right)^d .$$

Que se visualiza:

```
In [64]: pauli = PauliFeatureMap(feature_dimension=3, reps=1, paulis = ['X', 'Y', 'ZZ'])
         pauli.decompose().draw('mpl')
```

Out[64]:



Procedamos entonces crear una Matriz de Gram, que se utilizará a continuación para entrenar un SVM clásica.

```
zzgram_train = zz_kernel.evaluate(x_vec=entrenamiento)
zzgram_test = zz_kernel.evaluate(x_vec=prueba, y_vec=entrenamiento)
print(zzgram_train.shape)
print(zzgram_test.shape)
```

```
(100, 100)
(20, 100)
```

A la función de SVM de Sklearn se le puede indicar que se utilice un kernel analítico (como los que utilizamos en la subsección anterior) o se le puede indicar que utilice las entradas de una matriz con la función kernel ya calculada, es así como vamos a utilizar la Matriz de Gram.

In [67]:
```
zz_svc = SVC(kernel='precomputed')
zz_svc.fit(zzgram_train, entrenamiento_tag)
zz_score = zz_svc.score(zzgram_test, prueba_tag)


print(f'ZZ precisión: {zz_score}')
```

```
ZZ precisión: 0.7
```

En este caso, se obtuvo una precisión del 70 % utilizando el mapa de características ZZ. Utilizando otros mapas de características, como el mapa Z:

In [75]:
```
z_map = ZFeatureMap(feature_dimension=n_dim, reps=2)
z_kernel = QuantumKernel(feature_map=z_map, quantum_instance=Aer.get_backend('statevector_simulator'))
```

In [76]:
```
gram_train_z = z_kernel.evaluate(x_vec=entrenamiento)
gram_test_z = z_kernel.evaluate(x_vec=prueba, y_vec=entrenamiento)
```

In [81]:
```
pc_svc = SVC(kernel='precomputed')
pc_svc.fit(gram_train_z, entrenamiento_tag)
pc_score = pc_svc.score(gram_test_z, prueba_tag)

print(f'Z precisión: {pc_score}')
```

```
Z precisión: 0.8
```

Obteniéndose una precisión del 80 %, que en realidad es apenas igual a la precisión de los métodos clásicos, probablemente debido a que este set de datos en particular se separa linealmente más eficazmente con funciones kernel sencillas. En futuros estudios, se intentará buscar set de datos donde sí se logre alguna ventaja cuántica en términos de precisión.

0.4. Ventaja sobre el modelos clásico

Existen pocos problemas de clasificación en los cuales los metodos vistos representen una ventaja en comparación a los kernels clásicos, un ejemplo en el que se puede demostrar una mayor velocidad de resolución es donde nos basamos en el problema de logaritmo discreto, el cual no tiene manera eficiente de ser resuelto de forma clásica, pero puede ser resuelto por el algoritmo cuántico de Shor.

0.5. Conclusiones

Bibliografía

[1] Cortes, C., Vapnik, V. (1995) *Support-vector networks.* Springer: Mach Learn 20, 273–297. https://doi.org/10.1007/BF00994018

[2] Winston, P. (2010) *6.034 Artificial Intelligence. Fall 2010.* Massachusetts Institute of Technology: MIT OpenCourseWare, `https://ocw.mit.edu`. License: Creative Commons BY-NC-SA. (Consultado el: 10/10/2021)

[3] Carmona Juárez, E. J. (2016) *Tutorial sobre Máquinas de Vectores Soporte (SVM).* Departamento de Inteligencia Artificial: Universidad Nacional de Educación a Distancia.

[4] Sam Kirkiles. *Support Vector Machines From Scratch Part 3: The Optimization Objective.* `https://medium.com/samkirkiles/support-vector-machines-from-scratch-part-3-the-optimization-objective-85e6`

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

[6] Schuld, M., Killoran, N. (2019). Quantum machine learning in feature Hilbert spaces. Physical review letters, 122(4), 040504.

[7] Havlíček, V., Córcoles, A. D., Temme, K., Harrow, A. W., Kandala, A., Chow, J. M., Gambetta, J. M. (2019). Supervised learning with quantum-enhanced feature spaces. Nature, 567(7747), 209-212.

[8] IBM Quantum Team. (2021). 2021 Qiskit Global Summer School on Quantum Machine Learning. Qiskit. Recuperado 5 de enero de 2022, `https://qiskit.org/textbook-beta/summer-school/quantum-computing-and-quantum-learning-2021/`

[9] Heart Disease Data Set. Hungarian Institute of Cardiology. Budapest: Andras Janosi, M.D.

[10] Liu, Y., Arunachalam, S., Temme, K. (2021). A rigorous and robust quantum speed-up in supervised machine learning. Nature Physics, 17(9), 1013-1017.

[11] Qiskit tutorials. *Simulators.* Disponible en: `https://qiskit.org/documentation/tutorials/simulators/1_aer_provider.html`