Quantum Support Vector Machine

Roberto Esposito - 636672 - r.esposito8@studenti.unipi.it - MSc. in Artificial Intelligence

Introduction to Quantum Computing - 6 CFU - A.Y. 2021/2022

Abstract

In this report we will describe how a supervised learning technique, Support Vector Machine, can be improved using quantum computing. The Quantum SVM can be implemented mainly in three ways. The first is through the quantum variational classifier that works using a quantum variational circuit to classify the training set. The second way to implement a QSVM is thanks to the quantum kernel estimator that uses quantum computing to evaluate the kernel function. The last approach works only with quantum data and it is able to achieve an exponential speedup.

1 Introduction

Quantum Computing is a research field that is growing exponentially during the last years. Moreover some artificial intelligence techniques have been developed thanks to the quantum world because many operations of supervised and unsupervised learning can be optimized taking advantage of key points like entanglement and superposition.
In this report we analyze in details the Quantum Support Vector Machine in order to show how exponential improvements can be achieved regarding on its complexity, but first of all we need to introduce briefly the Quantum Machine Learning (QML).
When we talk about QML there are four different approaches that can be used to combine the quantum world with the machine learning. The different types are reported in the following Figure 1 and they are distinguished by two things: the type of data and the type of algorithm.
The first one is denoted as **CC**, which means that we have classical data (C) and classical computers (C), but the algorithms we use are inspired by quantum computing. The second approach is **CQ**, where the processed data are classical (C) and the machine learning algorithms are quantum (Q). On the other hand we can have **QC**, that works with quantum data (Q), but the machine learning algorithms are classical (C). The last approach is the less developed with respect to the others since it works either with quantum data (Q) either with quantum machine learning algorithms (Q) and it is called **QQ**. Among these four approaches, we will focus in particular on classical data and quantum algorithms (CQ) because most of these algorithms can be executed on near-term quantum devices.

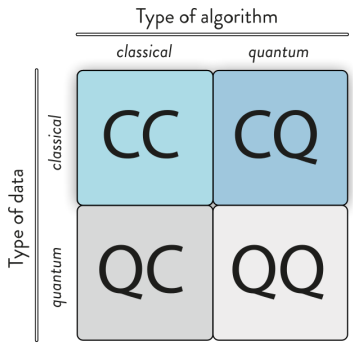

Figure 1: Quantum Machine Learning approaches.

The **Support Vector Machine** is a supervised machine learning algorithm used for classification problems.
Suppose we are given m training data from a training set T, where $|T| = m$. Each sample has the following form: $\{(\vec{x}_j, y_j) : \vec{x}_j \in \mathbb{R}^d, y_j = \pm 1\}$ where $y_j = +1$ or $y_j = -1$ depending on the class of $\vec{x}_j$. Our goal is to find a hyperplane ($\vec{w}$) which maximizes the margin between the two classes. The margin is defined by two parallel hyperplanes that are separated by the maximum possible distance without samples inside this margin (it is called **hard margin**); if we admit some data then it takes the name of **soft margin**. In the following Figure 2 it is reported an application of Support Vector Machine, where $\vec{x}_j \in \mathbb{R}^2$.

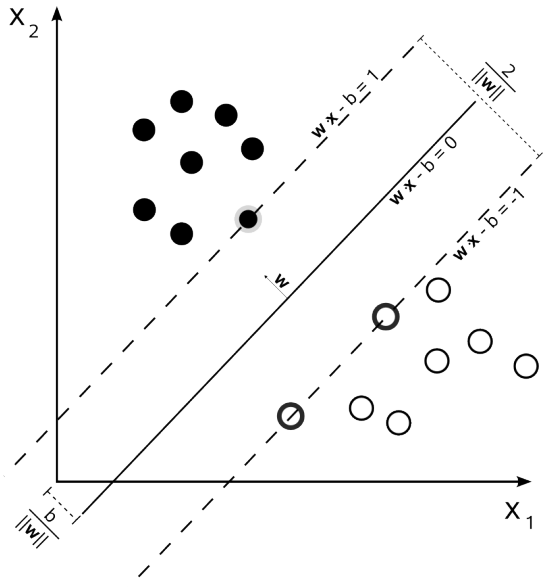

Figure 2: Support Vector Machine

The sample $\vec{x}_j$ belongs to the class $y_j = +1$ if $\vec{w} \cdot \vec{x_j} + b \geq 1$, or to $y_j = -1$ if $\vec{w} \cdot \vec{x_j} + b \leq -1$.
The optimization problem can be formulated as finding the optimum values of $\vec{w}$ and b which minimize:

$$\begin{aligned} \text{(P)} \quad & \min_{\vec{w},b} \quad \frac{1}{2}\vec{w} \cdot \vec{w} = \frac{|\vec{w}|^2}{2} \\ & \text{s.t.} \quad y_j(\vec{w} \cdot \vec{x}_j + b) \geq 1 \quad \forall j = 1, \ldots, m \end{aligned} \tag{1}$$

The dual formulation of the minimization problem reported above consists of maximizing over the Lagrange multipliers:

$$\begin{aligned} \text{(D)} \quad & \max_{\vec{\alpha}} \quad \sum_{j=1}^{m} y_j \alpha_j - \frac{1}{2} \sum_{j,k=1}^{m} \alpha_j K_{jk} \alpha_k \\ & \text{s.t.} \quad \sum_{j=1}^{M} \alpha_j = 0 \\ & \qquad y_j \alpha_j \geq 0 \quad \forall j = 1, \ldots, m \end{aligned} \tag{2}$$

After solving the dual problem (D), the optimal values are computed as: $\vec{w} = \sum_{j=1}^{m} \alpha_j \vec{x}_j$ and $b = y_j - \vec{w} \cdot \vec{x}_j$.
The value K_{jk} is an element of the **kernel matrix** that defines the kernel function $K_{jk} = k(\vec{x}_j, \vec{x}_k) = \vec{x}_j \cdot \vec{x}_k$. In order to obtain the optimal values to build the hyperplane we need to compute $m(m-1)$ dot products and then find the optimal values α_j using the quadratic programming.
Once we get the hyperplane, whenever we want to classify a new data $\vec{x}$, we just need to take:

$$y(\vec{x}) = \text{sign}\left(\sum_{j=1}^{m} \alpha_j k(\vec{x}_j, \vec{x}_k) + b \right)$$

In classical computation the SVM can be solved in polynomial time with respect to the number of training data (m), the dimension of the feature space (d) and the accuracy ϵ. At this point we can show how, using quantum computing, we can achieve an exponential speedup, reaching a logarithmic complexity.

2 Quantum Support Vector Machine

In this section we are going to analyze three ways to implement Quantum Support Vector Machine. Havlicek V. et al. [3] proposed two approaches: the first one uses the quantum variational circuits, while the second one uses the quantum computing only to estimate the kernel function, but the main algorithm is still the classical one. On the other hand, Rebentrost P. et al. [4] reaches a logarithmic complexity when the data are provided in coherent superposition, but if they are given in the classical way then this algorithm can not be applied.

2.1 Quantum Variational Classifier

The Quantum Variational Classifier works on a parameterized quantum circuit. Before we define how it is implemented, we should specify how the data are encoded.
The encoding consists in finding a way to encode classical data in quantum data in such a way that they can be processed by quantum system. There are several ways to encode them, like:

- **Basis encoding**: it encodes a string made of d-bit from a computational basis state in a system of d-qubit;
- **Amplitude encoding**: it encodes the data into the amplitude of a quantum state;
- **Angle encoding**: it encodes d features into the rotation angles of a number of qubits greater or equal to d;
- **Higher Order Encoding**: it is similar to angle encoding, but it uses a higher number of gates since it includes more features in a rotation.

One of the most popular feature map used is the ZFeatureMap that, for each qubit applies Hadamard gates and a rotation with respect to the corresponding feature, this block can be repeated several times increasing the order of the encoding. Another common feature map is the ZZFeatureMap that corresponds to the second order Pauli Z-evolution circuit and, compared to the previous one, it adds CNOT-gates and rotations with respect to more than one feature. In the following Figure 3 it is reported an example of ZZFeatureMap with 3 qubits, 1 repetition and linear entanglement.

Havlicek V. et al. [3] defines a feature map on d-qubits that is generated through the following unitary function: $\mathcal{U}_\Phi(\vec{x}) = U_{\Phi(\vec{x})} H^{\otimes n} U_{\Phi(\vec{x})} H^{\otimes n}$ and $U_{\Phi(\vec{x})}$ is the Pauli Expansion circuit defined as follows:

$$U_{\Phi(\vec{x})} = \exp\left(i \sum_{S \subset [n]} \phi_S(\vec{x}) \prod_{i \in S} Z_i \right)$$

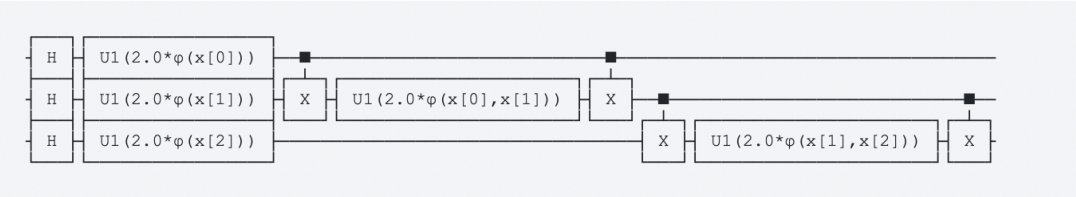

Figure 3: ZZFeatureMap.

where S is the test set, $\phi_S(\vec{x})$ is the encoded version of $\vec{x}$ and Z_i is a diagonal gate in the Z-basis. In the following Figure 4, it is reported the corresponding circuit. Notice that the unitary gate $U_\Phi(\vec{x})$ can be changed with another transformation, we just need it to be diagonal and implemented efficiently.

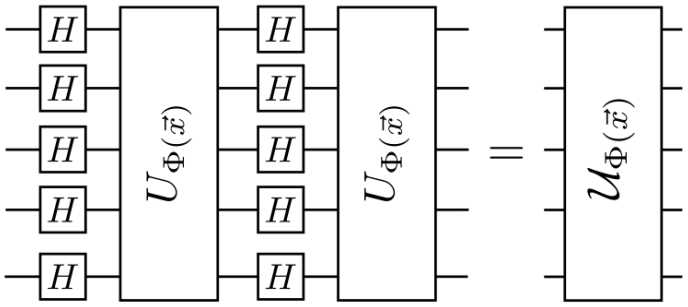

Figure 4: $\mathcal{U}_\Phi(\vec{x})$ Feature map.

Once the data are mapped to the feature space we give as input the feature vectors to the parameterized quantum circuit $W(\vec{\theta})$. If the variational circuit has l layers, then the parameter $\vec{\theta}$ belongs to $\mathbb{R}^{2d(l+1)}$ and its value will be optimized during the training state, obtaining the optimal value $\vec{\theta^*}$ (Figure 5).

The next step consists of measuring the quantum circuit in order to classify the data choosing its label $y \in \{+1, -1\}$; for this reason it is applied a binary measurement $\{M_y\}$ to the state $W(\vec{\theta^*})\, \mathcal{U}_{\Phi(\vec{x})} \left|0\right\rangle^n$.
The measurement with respect to the label y is defined as:

$$M_y = 2^{-1}(\mathbb{1} + y\mathbf{f})\,,$$

where $\mathbf{f} = \sum_{z\in\{0,1\}^n} f(z) \left|z\right\rangle \left\langle z\right|$. It follows that the probability to have the label y is $p_y(\vec{x}) = \left\langle \Phi(\vec{x})\right| W^H(\vec{\theta}) M_y W(\vec{\theta}) \left|\Phi(\vec{x})\right\rangle$. The measurement is repeated several times, obtaining the empirical distribution $\hat{p}_y(\vec{x})$, and it is assigned the label y to the data $\vec{x}$ whenever $\hat{p}_y(\vec{x}) > \hat{p}_{-y}(\vec{x}) - yb$, where $b \in [0, 1]$ is the bias.

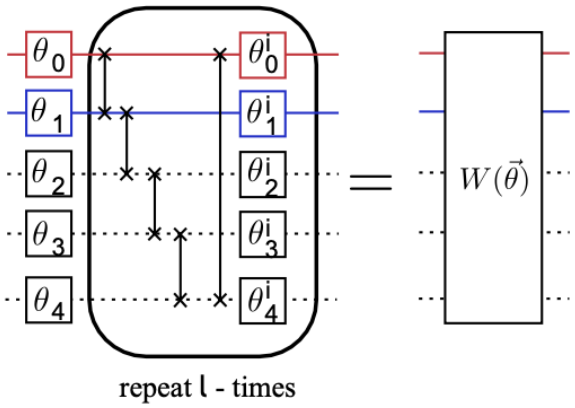

Figure 5: Variational circuit used to optimize the classification

At this point we are able to define the empirical risk that corresponds to the average of the probability of misclassifying the label of a sample over the whole training set:

$$R_{\text{emp}}(\vec{\theta}) = \frac{1}{|T|} \sum_{\vec{x} \in T} Pr(\tilde{m}(\vec{x}) \neq m(\vec{x}))$$

2.2 Quantum Kernel Estimator

In this section it is analyzed a second approach to implement Quantum Support Vector Machine, in particular this protocol uses the quantum computer twice because the main steps are done through the classical computers. The first time we use the quantum computing is to estimate the kernel matrix $K(\vec{x}_i, \vec{x}_j)$ $\forall \vec{x}_i, \vec{x}_j \in T = \{\vec{x}_1, \ldots, \vec{x}_m\}$.
The optimization problem defined in (2) can be restated in the following way:

$$\begin{aligned} \max \quad & L_D(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j K(\vec{x}_i, \vec{x}_j) \\ \text{s.t.} \quad & \sum_{i=1}^{m} \alpha_i y_i = 0 \\ & \alpha_i \geq 0 \quad \forall i = 1, \ldots, m \end{aligned} \tag{3}$$

Quantum computing is used a second time to estimate the kernel function whenever we need to classify the label of a new data $\vec{s} \in S$ (test set):

$$y_s = \text{sign}\left(\sum_{i=1}^{m} y_i \alpha_i^* K(\vec{x}_i, \vec{s}) + b \right)$$

where $\alpha*$ is the optimal vector containing all the α_i. Furthermore, this classification step is cheaper because most of the $\alpha_i^* = 0$ since only the support vectors (data points lying

on the separation hyperplane) will have a Lagrange multiplier strictly greater than 0. So, $K(\vec{x}_i, \vec{s})$ need to be estimated only for those $\alpha_i^* > 0$.

At this point we need to discuss on how the kernel matrix can be estimated with quantum computing. The kernel entries of the matrix K express the fidelity between two feature vectors and there exist several ways to estimate the similarity between two quantum states.

Havlicek V. et al. [3] computed the fidelity starting from the transition amplitude $|\langle\Phi(\vec{x})|\Phi(\vec{z})\rangle|^2 = |\langle 0^n|\mathcal{U}^H_{\Phi(\vec{x})}\,\mathcal{U}_{\Phi(\vec{z})}|0^n\rangle|^2$ and successively they measured the final state in the Z-basis several times to estimate the transition probability; the corresponding circuit is shown in Figure 6.

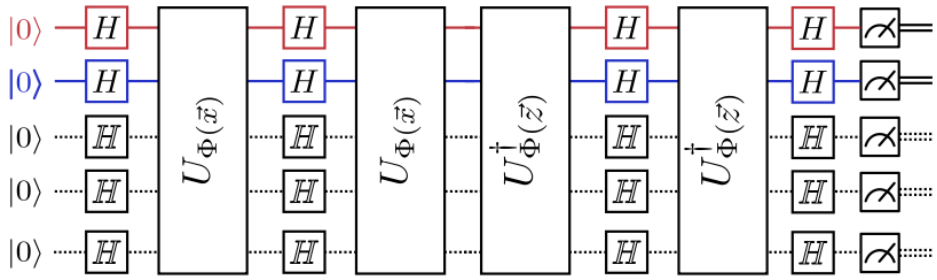

Figure 6: Circuit to estimate the fidelity between two feature vectors.

The resulting values obtained from this approach give, as output, values that are really close to the noise-free case.

The experiments have been run on three different training sets and they achieved 100% on all of them except for the third training set in which the reached accuracy is 94.75%.

If we look at the plot reported below, Figure 7(a), we can see on the left the kernel matrix obtained through the estimation and on the right the ideal kernel matrix; they are similar and the greater difference is in the 8^{th} row. In the Figure 7(b) are shown the differences of the entries of that row.

2.3 Logarithmic Quantum Support Vector Machine

In this section we are going to analyze another approach proposed by Rebentrost P. et al. [4], that achieves an exponential speedup improving the complexity from polynomial time down to logarithmic time with respect to the size of the training set and the dimension of the feature space.

Suppose we are already given the quantum data $|\vec{x}_j\rangle = \frac{1}{|\vec{x}_j|}\sum_{k=1}^{d}(\vec{x}_j)_k\,|k\rangle$, the norm $|\vec{x}_j|$ and the corresponding labels y_j, $\forall j = 1, \ldots, m$. These states are built thanks to the QRAM (Quantum RAM) that allows to store quantum data and requires only $O(MN)$

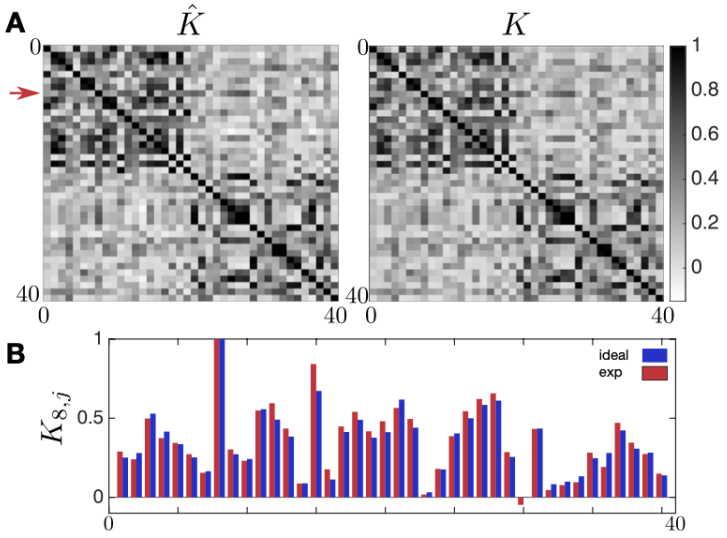

Figure 7: Comparison between the two kernel matrices.

hardware resources and $O(\log MN)$ access operations.
The key idea of this algorithm is to revise the problem definition of SVM and write it as a least-squares problem. This process is done using the slack variables, e_j, and replacing the inequality constraints with equality constraints:

$$y_j(\vec{w} \cdot \vec{x}_j + b) \geq 1 \rightarrow (\vec{w} \cdot \vec{x}_j + b) = y_j - y_j e_j \ .$$

The least-square approximation problem that comes out is the following one:

$$F \begin{bmatrix} b \\ \vec{\alpha} \end{bmatrix} \equiv \begin{bmatrix} 0 & \vec{1}^T \\ \vec{1}^T & K + \gamma^{-1}\mathbb{1} \end{bmatrix} \begin{bmatrix} b \\ \vec{\alpha} \end{bmatrix} = \begin{bmatrix} b \\ \vec{y} \end{bmatrix} \ , \tag{4}$$

where K is the kernel matrix, γ is a penalty term user-defined and $\vec{\alpha}$ is the unknown variable.
The parameters of the SVM are computed with the inverse of F: $(b, \vec{\alpha}^T) = F^{-1}(0, \vec{y}^T)^T$.
In the quantum case we solve the normalized $\hat{F}\,|b, \vec{\alpha}^T\rangle$, where $\hat{F} = F/\mathrm{tr}F$ and $||F|| \leq 1$.
The matrix $\hat{F}$ is decomposed as:

$$\hat{F} = (J + K + \gamma^{-1}\mathbb{1})/\mathrm{tr}F\,, \quad J = \begin{bmatrix} 0 & \vec{1}^T \\ \vec{1}^T & 0 \end{bmatrix} \ . \tag{5}$$

In general, when we have a quantum system, if we want to compute the inverse of a matrix (in this case $\hat{F}^{-1}$) to solve it, we need to compute $e^{-i\hat{F}\Delta t}$.
Recalling the decomposition (5) and what we have just said, using the Lie product formula we get:

$$e^{-i\hat{F}\Delta t} = e^{-i\Delta t\mathbb{1}/\mathrm{tr}F}\, e^{-iJ\Delta t/\mathrm{tr}F}\, e^{-iK\Delta t/\mathrm{tr}F} + O(\Delta t^2)$$

The right hand side of the equation (4) can be decomposed into the eigenstates of $\hat{F}$ with the corresponding eigenvalues and, following the steps proposed in [2] by Harrow A. to solve linear systems, we can get the SVM parameters:

$$|b, \vec{\alpha}\rangle = \frac{1}{\sqrt{b^2 + \sum_{k=1}^{m} \alpha_k^2}} \left(b\,|0\rangle + \sum_{k=1}^{m} \alpha_k\,|k\rangle \right)$$

At this point we have the parameters of the Quantum SVM and whenever we would like to classify a new data $|\vec{x}\rangle$ we need to construct the state to query the classifier:

$$|\vec{x}\rangle = \frac{1}{\sqrt{m|\vec{x}|^2 + 1}} \left(|0\rangle\,|0\rangle + \sum_{k=1}^{m} |\vec{x}|\,|k\rangle\,|\vec{x}\rangle \right)$$

Using an ancillary qubit, we get $|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle\,|\tilde{u}\rangle + |1\rangle\,|\tilde{x}\rangle)$ and then, we measure the ancillary qubit in the state $|-\rangle$. The measurement success has probability $P = |\langle\psi|\,|-\rangle|^2 = \frac{1}{2}(1 - \langle\tilde{u}\,|\tilde{x}\rangle|)$ and, if $P < \frac{1}{2}\,|x\rangle$, it is classified in the state $+1$ (-1 otherwise).
Usually support vector machine are more powerful when they are used for nonlinear classification problems, which consist of mapping the input data $\vec{x}_j$ in a higher dimensional feature space, obtaining $\vec{\phi}(\vec{x}_j)$; so the kernel entries become a product of nonlinear transformations. For example, it can be used the mapping $k(\vec{x}_j, \vec{x}_i) = (\vec{x_j} \cdot \vec{x_k})^r$ and, in order to do this, we just need to map the input data into a r-tensor product $|\phi(\vec{x}_j)\rangle = |\vec{x}_j\rangle \otimes \cdots \otimes |\vec{x}_j\rangle$. In this way, the kernel matrix can be computed efficiently and we still have a quantum advantage in computing the inner products.
The time complexity of this approach, as we said before, is $O(\log NM)$ and, in the nonlinear case, it is polynomial in the dimension r, so we have $O(r \log NM)$. In general, this approach (in linear case and nonlinear case) can be used only when our data are already in a quantum version and in a coherent superposition.

3 Experiments

In this section we show the results obtained with some experiments using the approach proposed by Havlicek V. et al. [3], in which the feature map is not computed with classical computing, but using a quantum process that allows a faster estimation of the feature space, in this case it is used the quantum simulator `qasm_simulator`.
First of all we need a dataset and for this reason we use the method `ad_hoc_data` that is built in such a way that the data can be separated achieving a 100% accuracy. In the following Figure 8 it is reported the dataset used during the experiments, that is made of 40 data where 30 of them belong to the training set and the others to the test set.

The next step requires to compute the quantum kernel matrix K and it is done using either the ZZFeatureMap either the ZFeatureMap with different number of repetitions

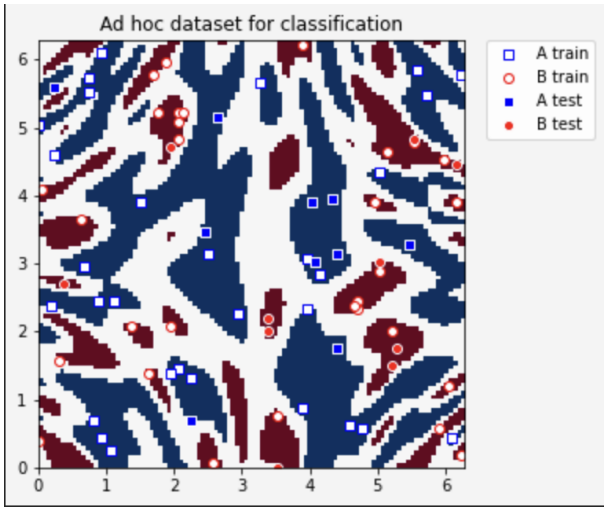

Figure 8: Dataset made of 40 data.

of the circuits. It turns out that increasing the number of `reps` does not ensure an improvement on the performance. In fact, the 100% accuracy is reached only using the ZZFeatureMap with two repetition of the circuit and a linear entanglement structure (Figure 9), but if we use `reps=3` or `reps=4` the accuracy goes down to 90%.

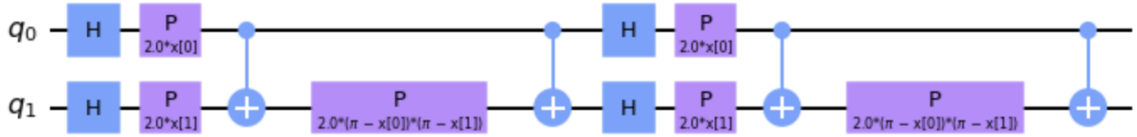

Figure 9: ZZFeatureMap.

As soon as the kernel function is defined we can build the QSVM thanks to the class `QSVC`, that extends the class of `sklearn.svm.SVC` inheriting the methods `fit()` and `predict()`.
Furthermore, we ran some tests using a different set (the `breast_cancer` dataset) and the performance degrade achieving a bad accuracy.

4 Conclusion

In this report we have introduced some techniques to build Quantum Support Vector Machine and it turns out that they can improve the time complexity with respect to the classical SVM if some conditions are satisfied.
For example, the first two approaches we proposed are useful only if the kernel function used by the classifier can not be computed with classical computers, otherwise it is not convenient to use quantum computing. The Quantum Variational Classifier achieves an accuracy close to 100% with an increasing depth of the circuit, while the Quantum Kernel Estimator is able to achieve a 100% accuracy.
In the third approach we have seen that it can be reached a logarithmic complexity in training size and feature dimension,; this is not the only advantage since we can ensure even data privacy because, as soon as the kernel matrix is generated, the features of the samples are hidden from the user.
The experiments were performed on an ad-hoc dataset built in such a way that allows us to achieve a 100% accuracy, but, if we use other datasets like the `breast_cancer` the performance decreases highly. All the experiments have been done using the open-source software `Qiskit` ([1]).

[1] Md Sajid Anis, Abby-Mitchell, Héctor Abraham, AduOffei, Rochisha Agarwal, Gabriele Agliardi, and Merav. Qiskit: An open-source framework for quantum computing, 2021.

[2] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical Review Letters*, 103(15), oct 2009.

[3] Temme K. Havlicek V., Corcoles A.D. Supervised learning with quantum-enhanced feature spaces. *Nature*, 567(7747):209–212, 2019.

[4] Patrick Rebentrost, M. Mohseni, and Seth Lloyd. Quantum support vector machine for big data classification. *Physical Review Letters*, 113, 07 2013.