

# Quantum Machine Learning

## Quantum Support Vector Machines

**Maximilian Forstenhäusler**

Chair of Scientific Computing in Computer Science  
TUM Department of Informatics  
Technical University of Munich

December 10<sup>th</sup>, 2021



*TUM Uhrenturm*

- 1 Introduction to Quantum Machine Learning
- 2 Support Vector Machines
- 3 Quantum Support Vector Machines
- 4 Results
- 5 References

# Introduction to Quantum Machine Learning

## Machine Learning

- Training machines to learn from the algorithms implemented to handle the data [1].
- Classical machine learning helps to classify images, recognize patterns and speech, handle big data, ...[1].
- Today, more and more data is being generated.  
→ Classical algorithms become less efficient [1].

# Introduction to Quantum Machine Learning

## Machine Learning

- Training machines to learn from the algorithms implemented to handle the data [1].
- Classical machine learning helps to classify images, recognize patterns and speech, handle big data, ...[1].
- Today, more and more data is being generated.  
→ Classical algorithms become less efficient [1].

⇒ **Need to find alternative methods**

# Introduction to Quantum Machine Learning

## Quantum Machine Learning

- Quantum Machine Learning (QML) is the intersection between quantum computing and machine learning.
- **Motivation:**
  - QML is expected to speed up the performance of ML programs through the use of quantum mechanical properties, i.e. entanglement, superposition.
  - Quantum speed-up in supervised machine learning has recently been shown by researchers of *IBM Quantum* and *University of California, Berkley* [2].
- **How does QML work in general?**
  - QML integrates quantum algorithms within machine learning programs.
    - ⇒ Data can be classified, sorted and analyzed using quantum algorithms on a quantum computer.

# Outline

- 1 Introduction to Quantum Machine Learning
- 2 Support Vector Machines
  - Linear Classification
  - Support Vectors
  - Kernel Methods
  - Limitations to SVM
- 3 Quantum Support Vector Machines
- 4 Results
- 5 References

# Linear Classification

## Intro

- Linear Classification, i.e. Perceptron, SVM, seeks to find an optimal separating hyperplane between two classes of data in a dataset such that, with high probability, all training examples of one class are found only on one side of the hyperplane [3].

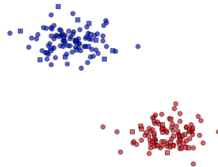
# Linear Classification

## Intro

- Linear Classification, i.e. Perceptron, SVM, seeks to find an optimal separating hyperplane between two classes of data in a dataset such that, with high probability, all training examples of one class are found only on one side of the hyperplane [3].

### Setup:

- Input vector  $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$   $x_i \in \mathbb{R}^D$
- Labels:  $\mathbf{y} = \{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_N\}$   $y_i \in \mathcal{C}$
- Classes:  $C = \{1, \dots, C\}$



**Figure 1** Linearly Separable Data.

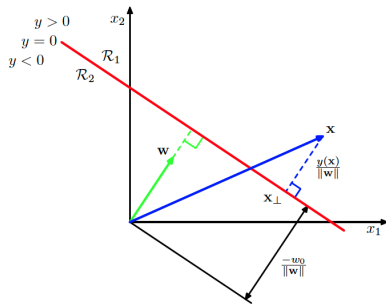


### Setup:

- Input vector  $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$   $x_i \in \mathbb{R}^D$
- Labels:  $\mathbf{y} = \{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_N\}$   $y_i \in \mathcal{C}$
- Classes:  $C = \{1, \dots, C\}$

### Find:

- $f(\cdot) : \mathbb{R}^D \rightarrow \mathcal{C}$
- In the case of a linear decision function:  
$$y(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + \mathbf{w}_0$$
- Famous perceptron algorithm [4]  
$$y_i(\mathbf{w}^T \phi(x_i) + \mathbf{w}_0) > 0$$



**Figure 1** Illustration of Hyperplane Decision Function [4].

# Linear Classification

## Limitations of LDA

1. No uncertainty measure
2. Hard to optimize
3. Poor generalization
4. Cannot handle noisy data

# Linear Classification

## Limitations of LDA

1. No uncertainty measure
2. Hard to optimize
3. Poor generalization
4. Cannot handle noisy data

**Extension:** Introduce variables that allow to maximize the margin of the hyperplane between the classes.

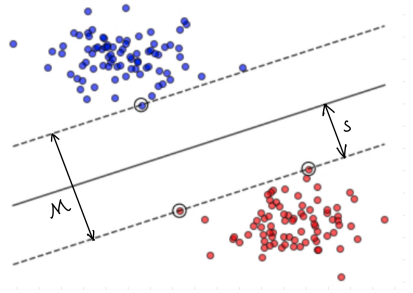
# Support Vectors

## Hard Margin - Introduction

- Add two hyperplanes to the decision function that are parallel to the decision function.
- Extend the additional hyperplanes until the first datapoint of a data cluster is reached.
- The distance between the two support hyperplanes is called the margin.

$$\mathcal{M} = \frac{2s}{||w||}$$

- A datapoint that is on the hyperplane is called a Support Vector.



**Figure 2** Illustration of Margin and Support Vectors.

# Support Vectors

## Hard Margin - Optimization Constraints

This setup leads to the following constraints, ( $s = 1$ ):

$$w^T x + b \geq +1 \text{ if } y = 1 \quad (1)$$

$$w^T x + b \geq -1 \text{ if } y = -1 \quad (2)$$

$\implies$  Maximize the margin  $\mathcal{M}$  (for mathematical convenience, minimize  $\mathcal{M} = \frac{1}{2}||w||^2$ ) given the above constraints [4].

$$\begin{aligned} \min f_0(\theta) \\ \text{s.t. } f_i(\theta) \geq 0 \text{ for } i = 1, \dots, N \end{aligned} \quad (3)$$

# Support Vectors

## Hard Margin - Optimization Constraints

This setup leads to the following constraints, ( $s = 1$ ):

$$w^T x + b \geq +1 \text{ if } y = 1 \quad (1)$$

$$w^T x + b \geq -1 \text{ if } y = -1 \quad (2)$$

$\implies$  Maximize the margin  $\mathcal{M}$  (for mathematical convenience, minimize  $\mathcal{M} = \frac{1}{2}||w||^2$ ) given the above constraints [4].

$$\begin{aligned} \min \quad & \frac{1}{2}||w||^2 \\ \text{s.t.} \quad & y_i(w^T x_i + b - 1) \geq 0 \text{ for } i = 1, \dots, N \end{aligned} \quad (3)$$

# Support Vectors

## Hard Margin - Quadratic Programming

This formulated optimization problem is a *quadratic programming* problem, which, in Python, can be solved with the CVXOPT library.

$$\begin{aligned} \min_x \quad & \frac{1}{2} \mathbf{x}^T \mathbf{P} \mathbf{x} + \mathbf{q}^T \mathbf{x} \\ \text{s.t.} \quad & \mathbf{G} \mathbf{x} \leq \mathbf{h}, \mathbf{A} \mathbf{x} = \mathbf{b} \end{aligned}$$

$\mathbf{q} \rightarrow N \times 1$  vector

$\mathbf{P} \rightarrow N \times N$  symmetric matrix

$\mathbf{G} \rightarrow N \times N$  diagonal matrix

$\mathbf{h} \rightarrow N \times 1$  vector

$\mathbf{A} \rightarrow M \times M$  matrix

$\mathbf{b} \rightarrow M \times 1$  vector

# Support Vectors

## Hard Margin - Lagrangian

In order to solve the optimization problem, eq. (3), we introduce the Lagrangian,

$$\mathcal{L}(\theta, \alpha) = f_0(\theta) - \alpha_i f_i(\theta) \quad (4)$$

and the Duality perspective.

The principle of Duality in optimization theory states that there are two perspectives of approaching an optimization problem [5]. The two approaches:

- ☐ **Primal**
- ☐ **Dual**



# Support Vectors

## Hard Margin - Lagrangian

In order to solve the optimization problem, eq. (3), we introduce the Lagrangian and the Duality perspective.

**Primal Problem:** Solves the lower bound of the constrained optimization ( $f_0(\theta^*) = p^*$ )

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2}w^T w - \sum_{i=1}^N \alpha_i (y_i (w^T x_i + b) - 1) \quad (5)$$

$$g(\alpha) = \min_{w, b} \mathcal{L}(w, b, \alpha) \quad (6)$$

$$\nabla_w \mathcal{L}(w, b, \alpha) = w - \sum_{i=1}^N \alpha_i y_i x_i \stackrel{!}{=} 0 \quad (7)$$

$$\nabla_b \mathcal{L}(w, b, \alpha) = - \sum_{i=1}^N \alpha_i y_i \stackrel{!}{=} 0 \quad (8)$$

# Support Vectors

## Hard Margin - Lagrangian

**Dual Problem:** The best lowest bound to the solution of the Primal problem ( $g(\alpha^*) = d^*$ )  
→ Substitute eq. (7) and eq. (8) back into the Lagrangian,  $\mathcal{L}(w^*, b^*, \alpha)$ .

$$g(\alpha) = \mathcal{L}(w^*, b^*, \alpha) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j x_i^T x_j \quad (9)$$

From here, set up a new constraint optimization problem:

$$\begin{aligned} & \max_{\alpha} g(\alpha) \\ & s.t. \sum_{i=1}^N \alpha_i y_i = 0, \quad \alpha_i \geq 0 \text{ for } i = 1, \dots, N \end{aligned} \quad (10)$$

# Support Vectors

## Hard Margin - Implementation

- With the constraint optimization problem, eq. (10), we can formulate a quadratic *programming problem* that can be implemented in Python with the CVXOPT library.

$$\begin{aligned} \max_{\alpha} \quad & \alpha \mathbf{1}_N - \frac{1}{2} \alpha^T \mathbf{Q} \alpha \quad \parallel \text{ where } \mathbf{Q} = \mathbf{y} \mathbf{y}^T \circ \mathbf{x}^T \mathbf{x} \\ \text{s.t.} \quad & \sum_{i=1}^N \alpha_i y_i = 0, \quad \alpha_i \geq 0 \text{ for } i = 1, \dots, N \end{aligned} \quad (11)$$

# Support Vectors

## Hard Margin - Implementation

- With the constraint optimization problem, eq. (10), we can formulate a quadratic *programming problem* that can be implemented in Python with the CVXOPT library.

$$\begin{aligned}
 \min_{\alpha} \quad & \frac{1}{2} \alpha^T \mathbf{P} \alpha - \mathbf{1}_N^T \alpha \\
 \text{s.t.} \quad & y^T \alpha = 0, \quad -\alpha_i \leq 0 \text{ for } i = 1, \dots, N
 \end{aligned} \tag{11}$$

$$\begin{aligned}
 \mathbf{q} &:= -\mathbf{1}_N \in \mathbb{R}^{Nx1} \\
 \mathbf{P} &:= \mathbf{Q} \in \mathbb{R}^{NxN} \\
 \mathbf{G} &:= -\text{diag}(\mathbf{1}_N) \in \mathbb{R}^{NxN} \\
 \mathbf{h} &:= \mathbf{0}_N \in \mathbb{R}^{Nx1} \\
 \mathbf{A} &:= \mathbf{y} \in \mathbb{R}^{Nx1} \\
 \mathbf{b} &:= \mathbf{0}_N \in \mathbb{R}
 \end{aligned}$$

# Support Vectors

## Hard Margin - Support Vectors

**Duality:** We use the duality gap between the Primal and Dual solution,  $p^* - d^*$ , to find the Support Vector (strong duality =  $p^* - d^* = 0$ ):

→ Strong duality holds if,

$$\alpha_i(y_i(w^T x_i + b) - 1) = 0$$

implying that given  $\alpha_i > 0$ , if

$$y_i(w^T x_i + b) = 1$$

⇒ **Support Vector**

# Support Vectors

## Hard Margin - Classification

### Parameters:

$$w^* = \sum_{i=1}^N \alpha_i y_i x_i = (\boldsymbol{\alpha} \cdot \mathbf{y})^T \mathbf{X} \quad (12)$$

$$b = y_i - w^T x_i \quad (13)$$

### Classification:

$$\mathbf{y}_{pred} = \text{sign}((w^*)^T \mathbf{x} + b) \quad (14)$$

# Support Vectors

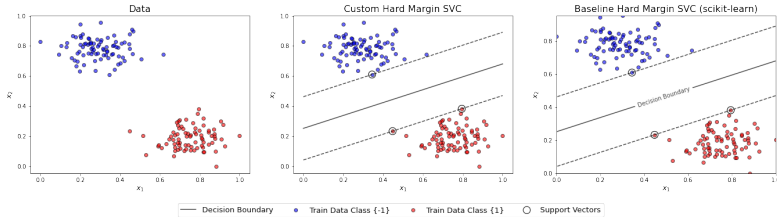
## Hard Margin - Implementation

Link to the implementation on GITHUB:

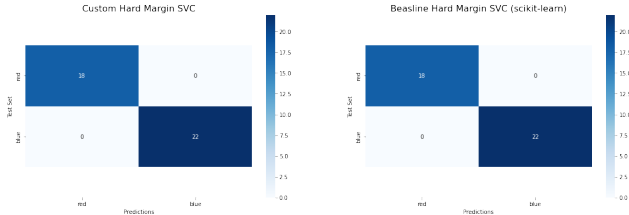
■ [linear-classifier](#)

# Support Vectors

## Hard Margin - Example Plot



**Figure 3** Custom Hard Margin SVC versus scikit-learn linear SVC.



**Figure 4** Confusion Matrices for custom hard margin SVC and SCIKIT baseline hard margin SVC.



# Support Vectors

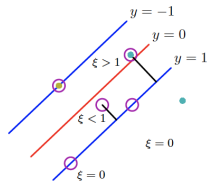
## Soft Margin

- Hard Margin classification assumes that the data is linearly separable, i.e. does not overlap.
- Unrealistic in real world problems.
- **Extend** the previous Support Vector method to allow misclassification .

# Support Vectors

## Soft Margin

- Hard Margin classification assumes that the data is linearly separable, i.e. does not overlap.
- Unrealistic in real world problems
- **Extend** the previous Support Vector method to allow misclassification.
- Introduce *slack variables*,  $\xi_i \geq 0$ , which measure the violation of the margin (in units of  $\|w\|$ ).



**Figure 5** Depiction of Slack Variables  $\xi$  [4].

$$\begin{aligned} \min f_0(\theta) \\ s.t. \quad f_i(\theta) \geq 0 \text{ for } i = 1, \dots, N \end{aligned} \tag{15}$$

$$\begin{aligned} \min \quad & \frac{1}{2} \|w\|_2^2 + C \sum_{i=1}^N \xi_i \\ \text{s.t.} \quad & y_i(w^T x_i + b - 1) \geq 1 - \xi_i \quad \forall_i \end{aligned} \tag{15}$$

$$\begin{aligned} \min \quad & \frac{1}{2} \|w\|_2^2 + C \sum_{i=1}^N \xi_i \\ \text{s.t.} \quad & y_i(w^T x_i + b - 1) \geq 1 - \xi_i \quad \forall_i \end{aligned} \tag{15}$$

Repeat the procedure:

1. Calculate the Primal
2. Calculate the Dual
3. Use Duality to determine support vectors and to calculate the decision function to perform the classification.

# Support Vectors

## Soft Margin - Primal

$$\mathcal{L}(w, b, \xi, \alpha, \mu) = \frac{1}{2} \|w\|_2^2 + C \sum_{i=1}^N \xi_i - \sum_{i=1}^N (\alpha_i (y_i (w^T x_i + b) - 1 + \xi_i) - \sum_{i=1}^N \mu_i \xi_i \quad (16)$$

$$g(\alpha) = \min_{w, b, \xi} \mathcal{L}(w, b, \xi, \alpha, \mu) \quad (17)$$

■ KKT conditions:

$$\alpha_i \geq 0 \quad (18) \qquad \mu_i \geq 0 \quad (21)$$

$$y_i (w^T x_i + b) - 1 + \xi_i \geq 0 \quad (19) \qquad \xi_i \geq 0 \quad (22)$$

$$\alpha_i (y_i (w^T x_i + b) - 1 + \xi_i) = 0 \quad (20) \qquad \mu_i \xi_i = 0 \quad (23)$$

■ Determine  $w^*$ ,  $b^*$  and  $\xi^*$

# Support Vectors

## Soft Margin - Dual

$$g(\alpha) = \mathcal{L}(w^*, b^*, \xi^*, \alpha) = \sum_{i=0}^N \alpha_i - \frac{1}{2} \sum_{i=0}^N \sum_{j=0}^N \alpha_i \alpha_j y_i y_j x_i^T x_j \quad (24)$$

$$\max g(\alpha) \quad (25)$$

$$s.t. \ 0 \leq \alpha_i \leq C \quad (26)$$

$$\sum_{i=1}^N \alpha_i y_i = 0 \quad \text{for } i = 1, \dots, N \quad (27)$$

■ Similarly, as with eq. (11), we reformulate eq. (24) as a *quadratic programming problem*.

# Support Vectors

## Soft Margin - Quadratic Programming

$$\min_{\alpha} \frac{1}{2} \alpha^T \mathbf{P} \alpha - \mathbf{1}_N^T \alpha \quad (28)$$

$$s.t. \ y^T \alpha = 0 \quad (29)$$

$$-\alpha_i \leq 0 \text{ for } i = 1, \dots, N \quad (30)$$

$$\alpha_i \leq C \text{ for } i = 1, \dots, N \quad (31)$$

$$\mathbf{q} := -\mathbf{1}_N \in \mathbb{R}^{Nx1}$$

$$\mathbf{P} := \mathbf{Q} \in \mathbb{R}^{NxN}$$

$$\mathbf{G} := - \begin{pmatrix} -\text{diag}(\mathbf{1}_N) \\ \text{diag}(\mathbf{1}_N) \end{pmatrix} \in \mathbb{R}^{2NxN}$$

$$\mathbf{h} := \begin{pmatrix} \mathbf{0}_N & C \cdot \mathbf{1}_N \end{pmatrix} \in \mathbb{R}^{2N \times 1}$$

$$\mathbf{A} := \mathbf{y} \in \mathbb{R}^{Nx1}$$

$$\mathbf{b} := \mathbf{0}_N \in \mathbb{R}$$



# Support Vectors

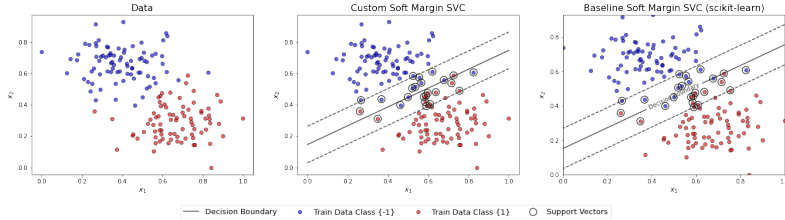
## Soft Margin Implementation

Link to the implementation on GITHUB:

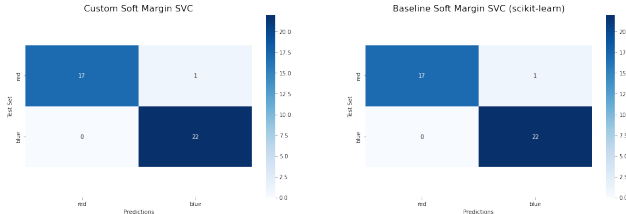
■ [linear-classifier](#)

# Support Vectors

## Soft Margin - Example Plot



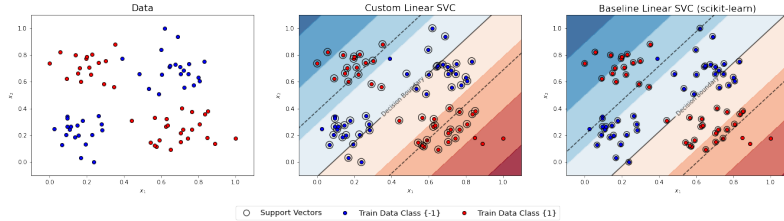
**Figure 6** Classification result of the custom soft margin SVC and SCIKIT baseline soft SVC.



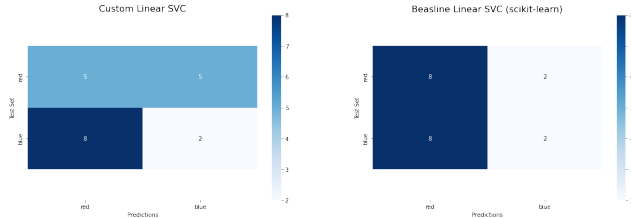
**Figure 7** Confusion Matrices for custom soft margin SVC and SCIKIT baseline soft margin SVC.

# Kernel Methods

## Soft Margin - Example Plot



**Figure 8** Displays the classification result of the custom soft margin SVC and SCIKIT baseline soft SVC.



**Figure 9** Confusion Matrices for custom soft margin SVC and SCIKIT baseline soft margin SVC.

## Kernel Methods

- So far, we still considered the data to be linearly separable in a feature space  $\mathbf{X}$ .
- In **extending** the Support Vector Machines for higher-dimensional data or non-linear data, we can further generalize the previous approaches by mapping d-dimensional data vectors into an n-dimensional feature space:

$$\phi : \mathbf{X} \rightarrow \mathcal{R}^n$$

- Further, we make use of the *Kernel Trick*:

$$-\frac{1}{2} \sum_{i=0}^N \sum_{j=0}^N \alpha_i \alpha_j y_i y_j \phi(x_i)^T \phi(x_j)$$

$$-\frac{1}{2} \sum_{i=0}^N \sum_{j=0}^N \alpha_i \alpha_j y_i y_j k(x_i, x_j), \quad \text{where } k(x_i, x_j) = \phi(x_i)^T \phi(x_j)$$

→ Increase computational speed for higher dimensional finite data.

## Kernel Methods

- So far, we still considered the data to be linearly separable in a feature space  $\mathbf{X}$ .
- In **extending** the Support Vector Machines for higher-dimensional data or non-linear data, we can further generalize the previous approaches by mapping d-dimensional data vectors into an n-dimensional feature space:

$$\phi : \mathbf{X} \rightarrow \mathcal{R}^n$$

- Further, we make use of the *Kernel Trick*. Therefore, eq. (24) can be written as,

$$g(\alpha) = \sum_{i=0}^N \alpha_i - \frac{1}{2} \sum_{i=0}^N \sum_{j=0}^N \alpha_i \alpha_j y_i y_j k(x_i, x_j) \quad (32)$$

- Kernel can be seen as a measure of similarity.

# Kernel Methods

**Linear:**

$$k(x_1, x_2) = x_1^T x_2$$

**Polynomial:**

$$k(x_1, x_2) = (c + x_1^T x_2)^d$$

**Radial Basis Function:**

$$k(x_1, x_2) = \exp(-\gamma \|x_1 - x_2\|^2)$$

**Sigmoid:**

$$k(x_1, x_2) = \tanh(\gamma x_1^T x_2 + c)$$

**Classification:** If the kernel is not linear, the classification function, eq. (14), must be adopted,

$$y_{pred} = \sum_{i \in SV} \alpha_i y_i k(x, x_i) \quad (33)$$

# Kernel Methods

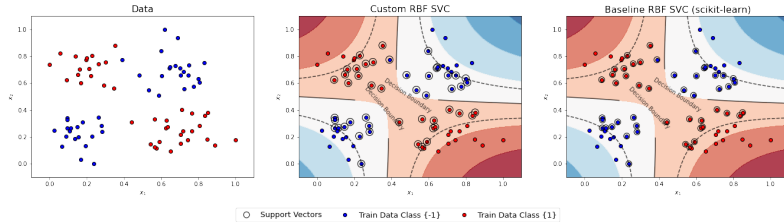
## Implementation

Link to the implementation on GITHUB:

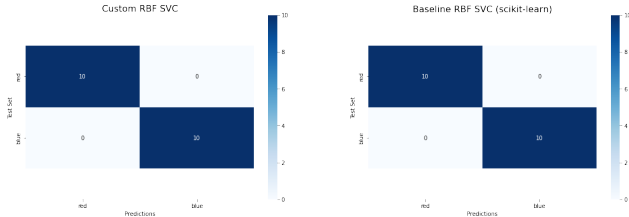
■ [nonlinear- classifier](#)

# Kernel Methods

## Kernel Method - Example Plot



**Figure 10** Displays the classification result of the custom RBF SVC and SCIKIT baseline RBF SVC.



**Figure 11** Confusion Matrices for custom RBF SVC and SCIKIT baseline RBF SVC.



# Limitations to SVM

- Feature space becomes large  $\implies$  Kernel functions become computationally expensive.
- No probabilistic interpretation of the classification.
- Not suitable for large datasets.

# Outline

- 1 Introduction to Quantum Machine Learning
- 2 Support Vector Machines
- 3 Quantum Support Vector Machines**
  - Quantum Kernel Estimation
  - Implementation of Quantum Support Vector Machines
- 4 Results
- 5 References

# Quantum Kernel Estimation

## Introduction

- Idea: Use the quantum advantage to speed up the computational speed of support vector classifiers (potential use of the exponentially large quantum state space).
- Two methods:
  1. *Quantum Variational Classifier*
  2. *Quantum Kernel Estimation*

# Quantum Kernel Estimation

## Introduction

- Idea: Use the quantum advantage to speed up the computation of support vector classifiers (potential use of the exponentially large quantum state space).
- Two methods:
  1. *Quantum Variational Classifier*
  2. *Quantum Kernel Estimation*
- In order to make use of the quantum advantage, the classical data needs to be transformed into the quantum state space.
  - ☐ Requires a data map (encoding function).
  - ☐ Requires a *quantum feature map* as a parameterized circuit.
- From the quantum state space, we can estimate a kernel matrix that can be used with a classical Support Vector Classifier.

# Quantum Kernel Estimation

## Quantum Feature Map

- Transforms low dimensional real space onto high dimensional quantum state space [6].

$$\Phi : \mathbf{x} \in \Omega \rightarrow |\Phi(x)\rangle \langle \Phi(x)| \quad [7] \quad (34)$$

- This is facilitated by a unitary operator  $\mathcal{U}_{\Phi(x)}$  on a initial state  $|0\rangle^n$  with  $n$  = number of qubits [8],

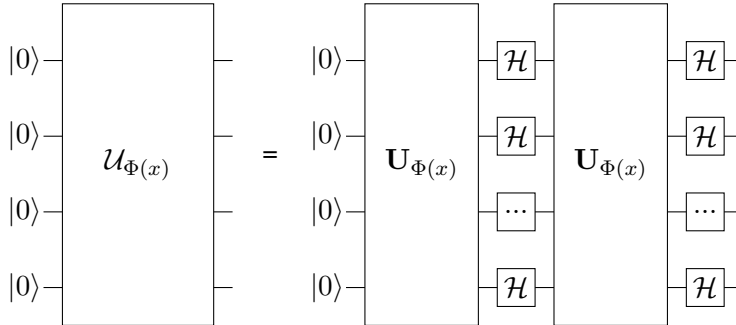
$$\Phi(x) = \mathcal{U}_{\Phi(x)} |0\rangle^{\otimes n} \quad (35)$$

$$\mathcal{U}_{\Phi(x)} = \prod_d \mathbf{U}_{\Phi(x)} \mathcal{H}^{\otimes n} \quad (36)$$

- Dimensions of the feature space have to align with the number of qubits.
- Feature maps have a high influence on the classification accuracy, as they are the basis for the kernel estimation  $\implies$  Careful analysis of the feature space is necessary [6].

# Quantum Kernel Estimation

## Quantum Feature Map - Quantum Circuit



# Quantum Kernel Estimation

## Pauli Feature Map

- The Pauli-Feature Map is a customizable example of a *quantum feature map* where the  $U_{\Phi(x)}$  from eq. (36) is a diagonal gate in the Pauli basis,

$$U_{\Phi(x)} = \exp \left\{ i \sum_{S \subseteq [n]} \phi_S(x) \prod_{k \in S} P_k \right\} \quad (37)$$

with

- $P_i \in \{I, \text{Pauli-X}, \text{Pauli-Y}, \text{Pauli-Z}\}$
- $S \in \left\{ \binom{n}{k} \text{ combinations}, k = 1, \dots, n \right\}$
- $\phi_S \rightarrow$  data mapping function (encoding function)
- Gives rise to the *Z-Feature Map* and the *ZZ-Feature Map*.

# Quantum Kernel Estimation

## Data Mapping functions

- Standard function used in QISKIT,

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

- Further examples [6]:

$$\phi_S : x \rightarrow \begin{cases} x_i & \text{if } S = \{i\} \\ \exp\left(\frac{|x_i - x_j|^2}{8/\ln(\pi)}\right) & \text{if } S = \{i, j\} \end{cases} \quad (38)$$

$$\phi_S : x \rightarrow \begin{cases} x_i & \text{if } S = \{i\} \\ \frac{\pi}{3 \cos x_i \cos x_j} & \text{if } S = \{i, j\} \end{cases} \quad (39)$$



# Quantum Kernel Estimation

## Quantum Feature Map - Example

■ Given eq. (37), if  $k = 2$ ,  $P_0 = Z$  and  $P_1 = ZZ \implies ZZ$ -Feature Map:

$$\mathbf{U}_{\Phi(x)} = \exp \left\{ \left( i \sum_{jk} \phi_S(j, k) Z_j \otimes Z_k \right) \left( i \sum_j \phi_S(j) Z_j \right) \right\} \quad (40)$$

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

$$\mathcal{U}_{\Phi(x)} = (\exp (ix_0 Z_0 + ix_1 Z_1 + i(x_0 - \pi)(x_1 - \pi) Z_0 Z_1) \mathcal{H}^{\otimes n})^d \quad (41)$$

# Quantum Kernel Estimation

## Quantum Kernel

- *Quantum feature maps*  $\Phi(x)$  naturally give rise to *quantum kernels*:

$$k(x_i, x_j) = \Phi(x_i)^\dagger \Phi(x_j) \quad (42)$$

- As the kernel entries are the fidelities between two feature vectors, we need to establish a way to estimate the fidelities of a quantum state.
- For finite data, this can be achieved by estimating the transition amplitude [7]:

$$K_{ij} = |\langle \Phi(x_i)^\dagger | \Phi(x_j) \rangle|^2 \quad (43)$$

- Plugging in eq. (35) into eq. (43):

$$K_{ij} = |\langle 0|^{\otimes n} \mathcal{U}_{\Phi(x)}^\dagger \mathcal{U}_{\Phi(x)} |0\rangle^{\otimes n}|^2 \quad (44)$$

# Quantum Kernel Estimation

## Quantum Kernel

- *Quantum feature maps*  $\Phi(x)$  naturally give rise to *quantum kernels*:

$$k(x_i, x_j) = \Phi(x_i)^\dagger \Phi(x_j) \quad (42)$$

- As the kernel entries are the fidelities between two feature vectors, we need to establish a way to estimate the fidelities of a quantum state.
- For finite data, this can be achieved by estimating the transition amplitude [7]:

$$K_{ij} = |\langle \Phi(x_i)^\dagger | \Phi(x_j) \rangle|^2 \quad (43)$$

- Plugging in eq. (35) into eq. (43):

$$K_{ij} = |\langle 0|^{\otimes n} \mathcal{U}_{\Phi(x)}^\dagger \mathcal{U}_{\Phi(x)} |0\rangle^{\otimes n}|^2 \quad (44)$$

$\implies$  *Quantum kernel matrix estimate*

# Implementation of Quantum Support Vector Machines

## Quantum Kernel Estimation



# Implementation of Quantum Support Vector Machines

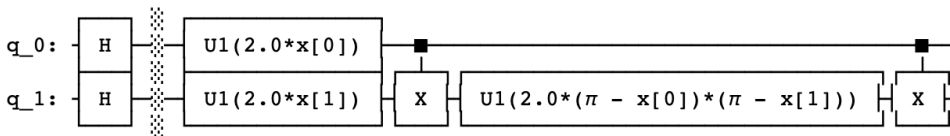
## Quantum Kernel Estimation

1. Build a parameterized quantum circuit that emulates a *quantum feature map* for each data pair.

*Example: ZZ-Feature Map* for 2-dimensional input

**Recall:** eq. (41)

$$\mathcal{U}_{\Phi(x)} = (\exp(ix_0 Z_0 + ix_1 Z_1 + i(x_0 - \pi)(x_1 - \pi)Z_0 Z_1) \mathcal{H}^{\otimes n})^d$$



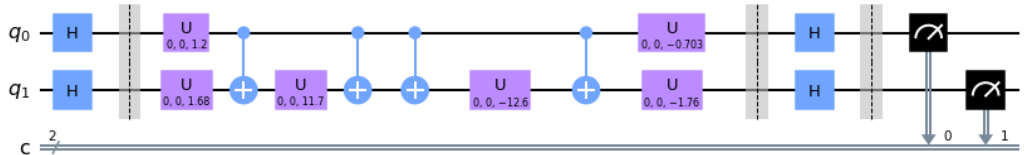
**Figure 12** ZZ-Feature Map Circuit.

# Implementation of Quantum Support Vector Machines

## Quantum Kernel Estimation

1. Build a parameterized quantum circuit that emulates a *quantum feature map* for each data pair.
2. **Construct the *quantum kernel* circuits for each data pair.**  
**Recall:** eq. (44) and eq. (41)

$$K_{ij} = |\langle 0|^{\otimes n} \mathcal{U}_{\Phi(x)}^\dagger \mathcal{U}_{\Phi(x)} |0\rangle^{\otimes n}|^2$$



**Figure 13** Parameterized Quantum Kernel Circuit.

# Implementation of Quantum Support Vector Machines

## Quantum Kernel Estimation



1. Build a parameterized quantum circuit that emulates a *quantum feature map* for each data pair.
2. Construct the *quantum kernel* circuits for each data pair.
3. **Measure the number of all zero strings  $0^n$ .**

# Implementation of Quantum Support Vector Machines

## Quantum Kernel Estimation

1. Build a parameterized quantum circuit that emulates a *quantum feature map* each data pair.
2. Construct the *quantum kernel* circuits for each data pair.
3. Measure the number of all zero strings  $0^n$ .
4. **Calculate the frequency of the zero strings to find the transition probability  $\implies$  Kernel entry of the the *quantum kernel*.**

**Note\*\*:** Step 3. and 4. compute the kernel values from the results of the inner products based on the measurements of the quantum circuits created for the *quantum kernel estimate*.



# Implementation of Quantum Support Vector Machines

## Quantum Kernel Estimation

1. Build a parameterized quantum circuit that emulates a *quantum feature map* each data pair.
2. Construct the *quantum kernel* circuits for each data pair.
3. Measure the number of all zero strings  $0^n$ .
4. Calculate the frequency of the zero strings to find the transition probability  $\implies$  Kernel entry of the the *quantum kernel*.

*Example:*

- Given a dataset  $\mathbf{X} \in \mathbb{R}^{10 \times 2}$
- Build parameterized kernel circuit from parameterized feature map circuits for each data pair  $\implies$  100 circuits.

# Implementation of Quantum Support Vector Machines

## Quantum Feature Map and Quantum Kernel Estimation

Links to the implementation on GITHUB:

- [quantum-feature-map](#)
- [quantum-kernel-estimation](#)

# Implementation of Quantum Support Vector Machines

## Quantum SVC



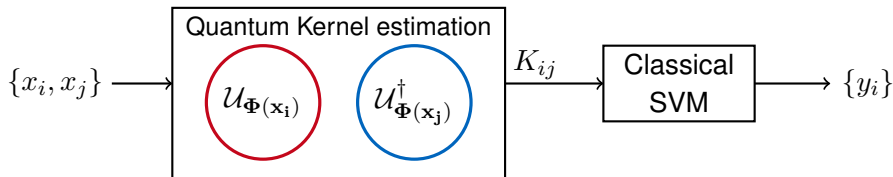
**How is the Quantum Kernel embedded into the SVM protocol?**

# Implementation of Quantum Support Vector Machines

## Quantum SVC

### How is the Quantum Kernel embedded into the SVM protocol?

- Instead of using a classical kernel in the constraint optimization problem, eq. (25), just insert the quantum kernel estimate, eq. (44).

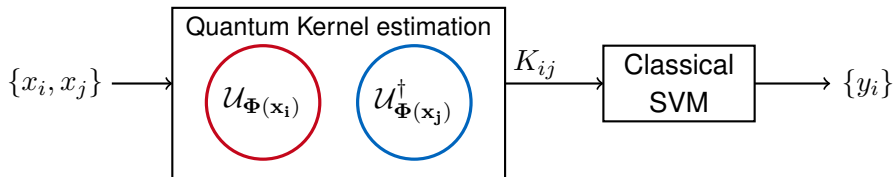


# Implementation of Quantum Support Vector Machines

## Quantum SVC

### How is the Quantum Kernel embedded into the SVM protocol?

- Instead of using a classical kernel in the constraint optimization problem, eq. (25), just insert the quantum kernel estimate, eq. (44).



- Now we, have a **Quantum Support Vector Machine**.

# Outline

- 1 Introduction to Quantum Machine Learning
- 2 Support Vector Machines
- 3 Quantum Support Vector Machines
- 4 Results**
- 5 References

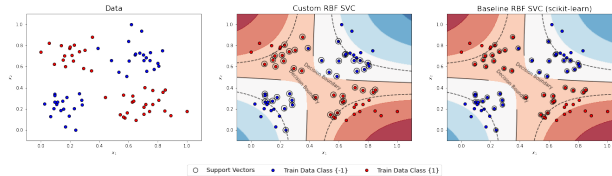
# Results

## Benchmark SVC

	Linear	Polynomial	Sigmoid	RBF C=10	RBF C=100		Linear	Polynomial	Sigmoid	RBF C=10	RBF C=100
<b>Linearly Separately Data</b>	0.975	0.975	0.975	0.975	0.975	<b>Linearly Separately Data</b>	0.975	0.975	0.95	0.975	0.975
<b>XOR Data</b>	0.350	0.950	0.400	1.000	1.000	<b>XOR Data</b>	0.500	0.950	0.35	1.000	1.000
<b>Circles Data</b>	0.400	1.000	0.275	1.000	1.000	<b>Circles Data</b>	0.400	0.800	0.30	1.000	1.000
<b>Moons Data</b>	0.750	0.750	0.750	0.750	0.850	<b>Moons Data</b>	0.750	0.725	0.75	0.725	0.875
<b>Adhoc Data</b>	0.300	0.250	0.300	0.500	0.550	<b>Adhoc Data</b>	0.350	0.300	0.40	0.500	0.550

(a) Test accuracy Custom SVC.

(b) Test accuracy benchmark SVC.

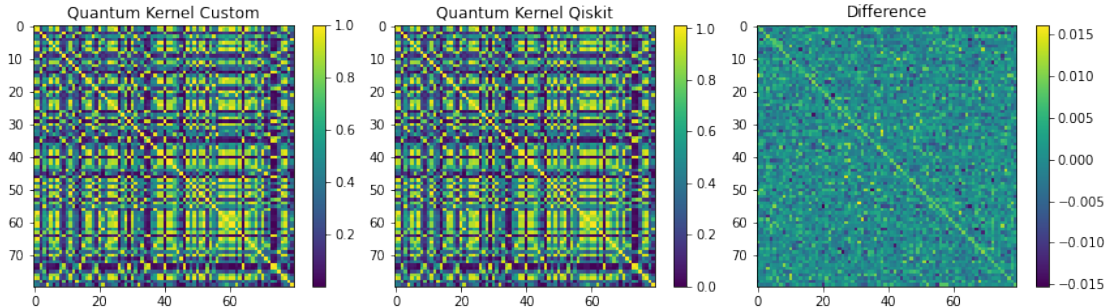


(c) Example plot on XOR data.

**Figure 14** Benchmark SVC.

# Results

## Benchmark Quantum Kernel

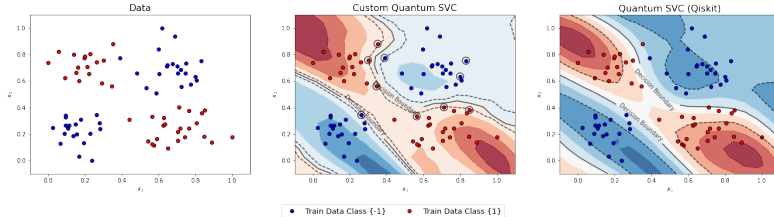


**Figure 15** Comparison of Custom Quantum Kernel and QISKIT quantum kernel.

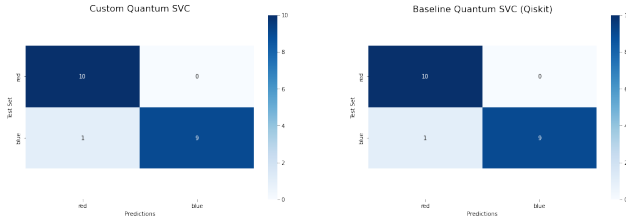


# Results

## Benchmark Quantum SVC



**Figure 16** Comparison of Custom Quantum SVC and QISKIT baseline Quantum SVC.



**Figure 17** Confusion Matrices for custom Quantum SVC and SCIKIT baseline Quantum SVC.

# Results

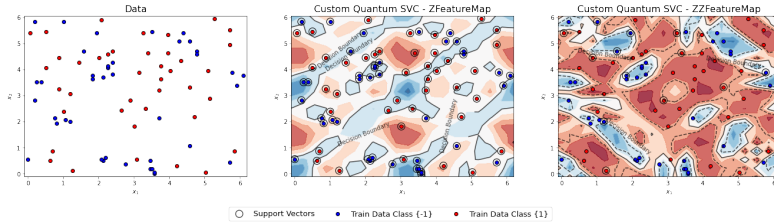
## Comparison of Different Data Maps in QSVC

	Default Data Map	Exp Data Map	Sin Data Map	Cos Data Map
<b>XOR Data</b>	0.9375	1.000	1.0000	1.0000
<b>Circles Data</b>	1.0000	1.000	1.0000	1.0000
<b>Moons Data</b>	0.6875	0.625	0.8125	0.8125
<b>Adhoc Data</b>	1.0000	0.500	0.7500	0.7500

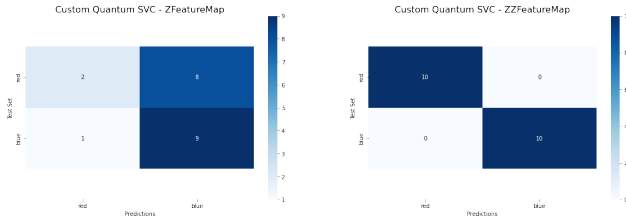
**Figure 18** Test accuracy of different data maps on different data.

# Results

## Comparison of the Z-Feature Map and the ZZ-Feature Map



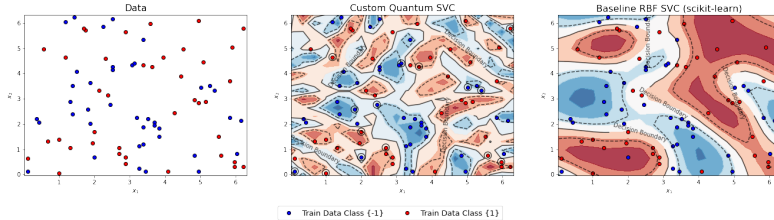
**Figure 19** Comparison of the Z-Feature Map and the ZZ-Feature Map on the Adhoc data [7].



**Figure 20** Confusion Matrix of Z-Feature Map and ZZ-Feature Map on the Adhoc data [7].

# Results

## Comparison of a Quantum Kernel and a RBF Kernel on Adhoc Data



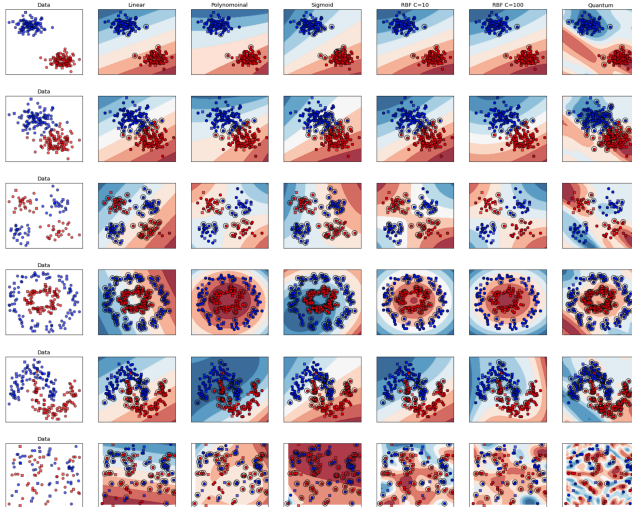
**Figure 21** Benchmark on Ad-hoc data.

	Custom Quantum SVC	Custom RBF SVC
<b>Adhoc Data</b>	1.0	0.7

**Figure 22** Test accuracy of custom QSVC and custom rbf SVC.

# Results

## Comparison of Different Kernels on Different Data



**Figure 23** Comparison of different classification models.

# Results

## Comparison of Different Kernels on Different Data

	Linear	Polynomoinal	Sigmoid	RBF C=10	RBF C=100	Quantum
<b>Linearly Separately Data</b>	0.975	0.975	0.975	0.975	0.975	0.950
<b>XOR Data</b>	0.350	0.950	0.400	1.000	1.000	0.950
<b>Circles Data</b>	0.400	1.000	0.275	1.000	1.000	0.975
<b>Moons Data</b>	0.750	0.750	0.750	0.750	0.850	0.850
<b>Adhoc Data</b>	0.300	0.250	0.300	0.500	0.550	1.000

**Figure 24** Comparison of different classification models.

# Results

## Conclusion

- Implemented a classical Support Vector Machine.
- Implemented a Quantum Kernel Estimation.
- Implemented a Quantum Support Vector Classifier.
- Benchmarked the custom implementations against implementations of established libraries.
- Showed that quantum kernels can perform better on high dimensional data.






### ■ Potential Improvements:

- ☐ Improve efficiency of code:
  - Remove unnecessary for-loops
  - Mirror quantum kernel in estimation as  $K_{ij} = K_{ji}$
  - Initialize the diagonal elements as 1
- ☐ Implement a general Pauli-Feature Map.
- ☐ Implement a state vector calculation scheme for the quantum kernel estimation.

- 1 Introduction to Quantum Machine Learning
- 2 Support Vector Machines
- 3 Quantum Support Vector Machines
- 4 Results
- 5 References**



# References I

-  N. Mishra, M. Kapil, H. Rakesh, A. Anand, N. Mishra, A. Warke, S. Sarkar, S. Dutta, S. Gupta, A. P. Dash, *et al.*, “Quantum machine learning: A review and current status”, *Data Management, Analytics and Innovation*, pp. 101–145, 2021.
-  Y. Liu, S. Arunachalam, and K. Temme, “A rigorous and robust quantum speed-up in supervised machine learning”, *Nature Physics*, vol. 17, no. 9, pp. 1013–1017, 2021.
-  J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, “Quantum machine learning”, *Nature*, vol. 549, no. 7671, pp. 195–202, 2017.
-  C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.
-  S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

## References II



Y. Suzuki, H. Yano, Q. Gao, S. Uno, T. Tanaka, M. Akiyama, and N. Yamamoto, “Analysis and synthesis of feature map for kernel-based quantum classifier”, *Quantum Machine Intelligence*, vol. 2, no. 1, pp. 1–9, 2020.



V. Havlíček, A. D. Córcoles, K. Temme, A. W. Harrow, A. Kandala, J. M. Chow, and J. M. Gambetta, “Supervised learning with quantum-enhanced feature spaces”, *Nature*, vol. 567, no. 7747, pp. 209–212, 2019.



A. Phan. (2021), 2021 qiskit global summer school on quantum machine learning - lab 3: Introduction to quantum kernels and svms, [Online]. Available: <https://learn.qiskit.org/summer-school/2021/lab3-introduction-quantum-kernels-support-vector-machines> (visited on 11/29/2021).