Quantum Computing Applications and use-cases: Quantum Kernel Methods

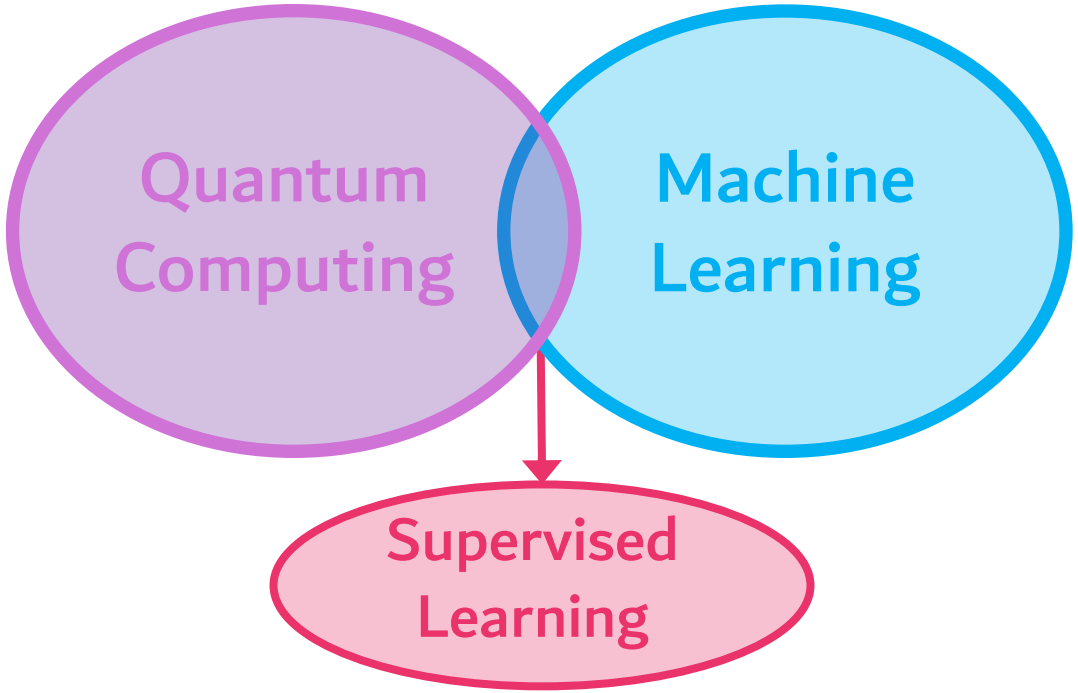
CERN openlab Summer Student Lecture Programme

R. Wixinger
ETH Zurich

F. Di Marcantonio
UPV/EHU Quantum Center
QTI CERN

Quantum Technology Initiative
01.08.2023

Quantum Machine Learning



Classification

Dataset and Estimators

- **Dataset (supervised)**

$$\mathcal{D} = \{ (\vec{x}_i, y_i) \quad \text{for} \quad i = 1, 2, \dots, N \}$$

$$\vec{x}_i = (x_i^1, x_i^2, \dots, x_i^p) \in \mathcal{X} \subseteq \mathbb{R}^p$$

BINARY LABEL:

$$y_i = \{-1, 1\}, \{0, 1\}, \{\mathrm{B}, \mathrm{R}\} \in Y \subseteq \mathbb{R}$$

↑ REGRESSION PROBLEM

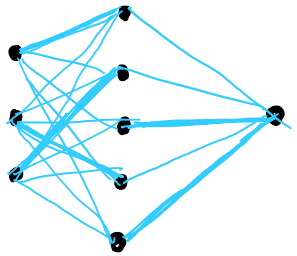
$$\tilde{y}_i = f(\vec{x}_i)$$

- **Estimators (basis functions)**

$$f : \mathcal{X} \subseteq \mathbb{R}^p \longrightarrow Y \subseteq \mathbb{R}$$

e.g. NN

x_i $g(x_i)$ $\tilde{y}_i$

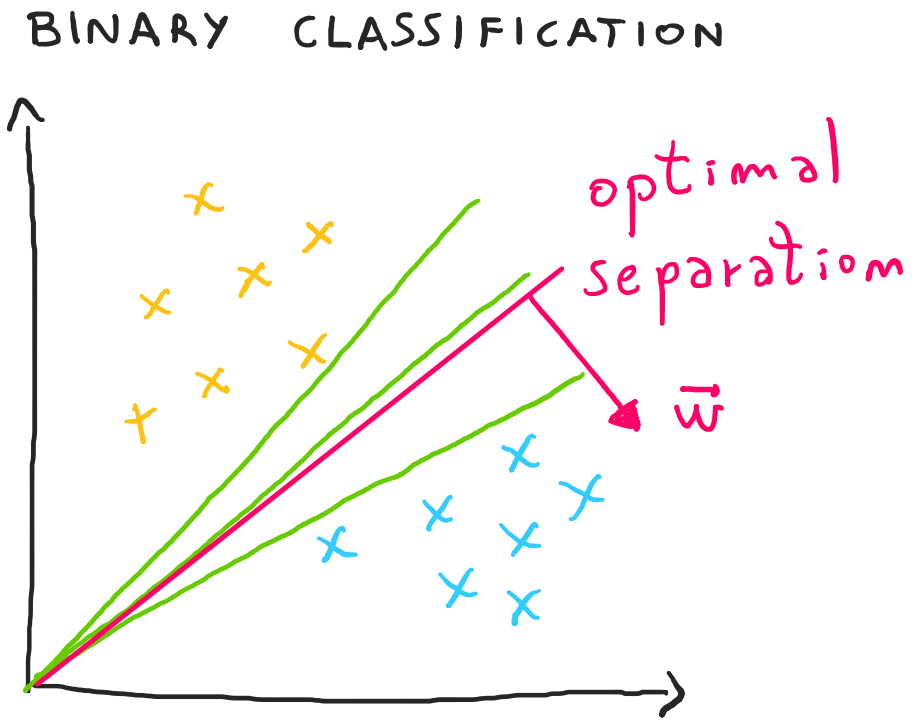


$$f : \mathcal{X} \to Z \longrightarrow Y$$

BASIS FUCTIONS

$$\sum_i \theta_i \, g(x_i, \phi_i) + \theta_0$$

Linear Classification Problem



$y_1 \; y_2 \; \dots \; y_N$

$\uparrow \; \uparrow \; \dots \; \uparrow$

Training $(\vec{x}_1, \vec{x}_2, \dots, \vec{x}_N)$

Dataset

Test $(\vec{s}_1, \vec{s}_2, \dots, \vec{s}_M)$

x_i^p, $i = 1, 2, \dots, N$ (# datapoint) $\quad p = 1, 2, \dots, m$ (# features)

$\hat{y} = \vec{w}^T \vec{x} + b \longrightarrow y \in [-1, 1]$

$\hat{y} = \text{label}(s) = \text{sign}(\vec{w}^T \vec{s} + b)$

Primal Form

Support Vector Machine

Constrained Primal Problem

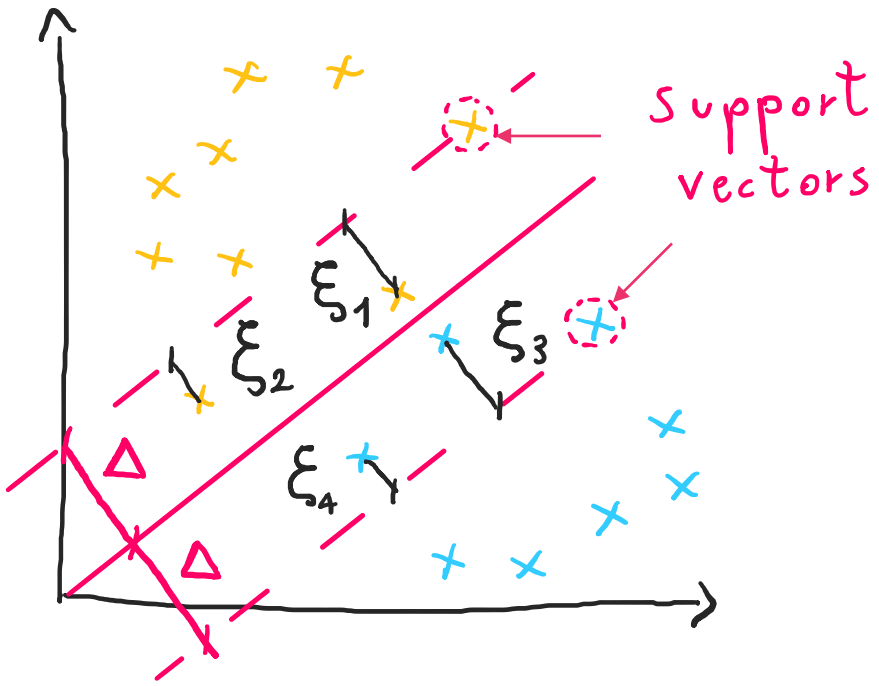
$$\min \frac{1}{2}\|\vec{w}\|^2$$

$$y_i[\vec{w}\cdot\vec{x}_i + b] \geq 1$$

NORMALIZATION

$$\|w\|\Delta = 1$$

BINARY CLASSIFICATION



$$\min_{i=1,2,\dots,m} y_i \frac{[\vec{w}\cdot\vec{x}_i + b]}{\|w\|} \geq \Delta$$

Hard

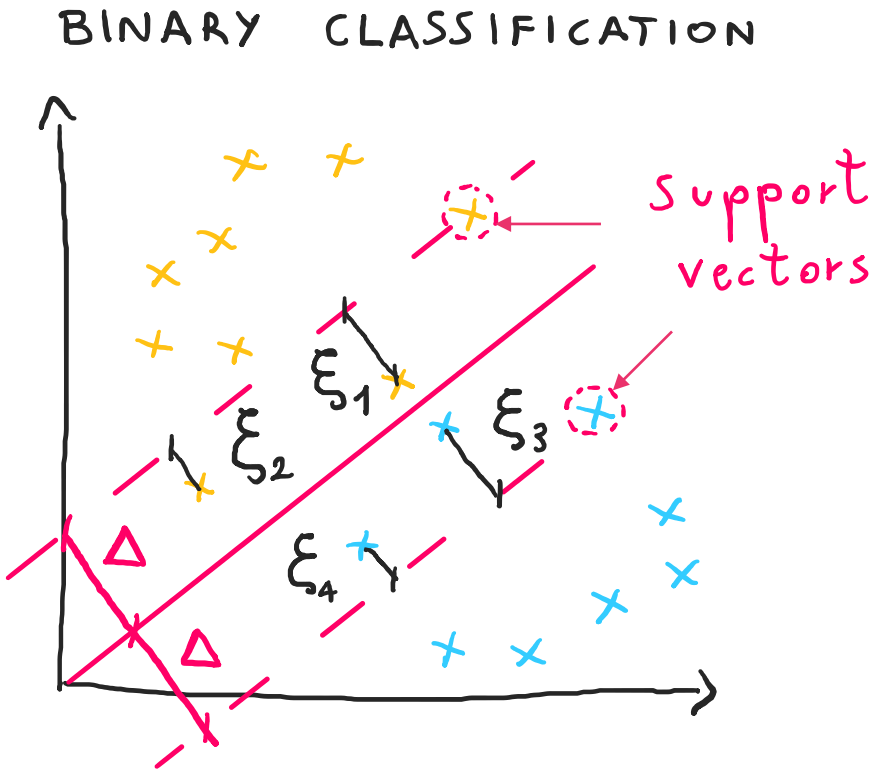
$$\min_{\vec{w},b,\xi} \quad \frac{1}{2}\|\vec{w}\|^2 + \frac{C}{m}\sum_i \xi_i$$

$$y_i[\vec{w}\cdot\vec{x}_i + b] \geq 1 - \xi_i$$

$$\forall i \in \{1, 2, \dots, N\}$$

Soft

Support Vector Machine



$$C = \frac{1}{2\lambda} \quad \text{regularization parameter}$$

$$\min_{\vec{w}, b, \xi} \quad \frac{1}{2}\|\vec{w}\|^2 + \frac{C}{m}\sum_i \xi_i$$

$$y_i\left[\vec{w}\cdot\vec{x}_i + b\right] \geq 1 - \xi_i$$

$$\forall i \in \{1, 2, \ldots, N\}$$

Soft

Dual (Lagrange) formulation and kernel trick

- **Primal problem with conditions:**

$$\min \frac{1}{2}\|w\|^2 \qquad y_i[\vec{w}\cdot\vec{x}_i+b]-1 \geq 0$$

- **Lagrange formula and Khun-Tucker conditions:**

$$\mathcal{L}(x_i,\alpha,w,b) = \frac{1}{2}\vec{w}\cdot\vec{w} + \sum_i \alpha_i\left[y_i(\vec{w}\cdot\vec{x}_i+b)-1\right]$$

$$\partial_\alpha\mathcal{L}, \partial_w\mathcal{L}, \partial_b\mathcal{L} = 0$$

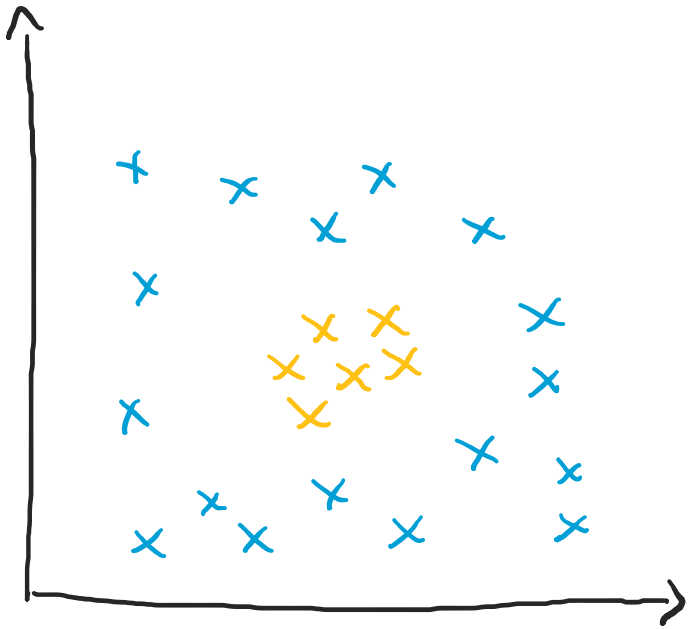
$$\alpha_i\cdot\left[y_i(\vec{w}\cdot\vec{x}_i+b)-1\right]=0 \begin{cases} \alpha_i = 0 \\ x_i \text{ is a support vector} \end{cases}$$

- **Final formula:**

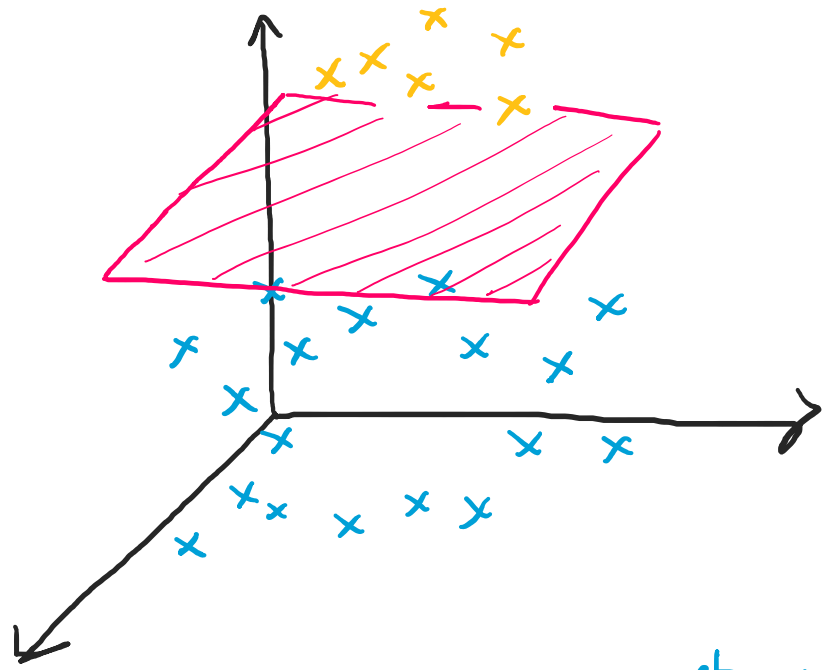
DUAL FORMULA

$$\mathcal{L} = -\frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j (\vec{x}_i\cdot\vec{x}_j)$$

Kernel Methods



$\phi(x)$



Not linearly separable

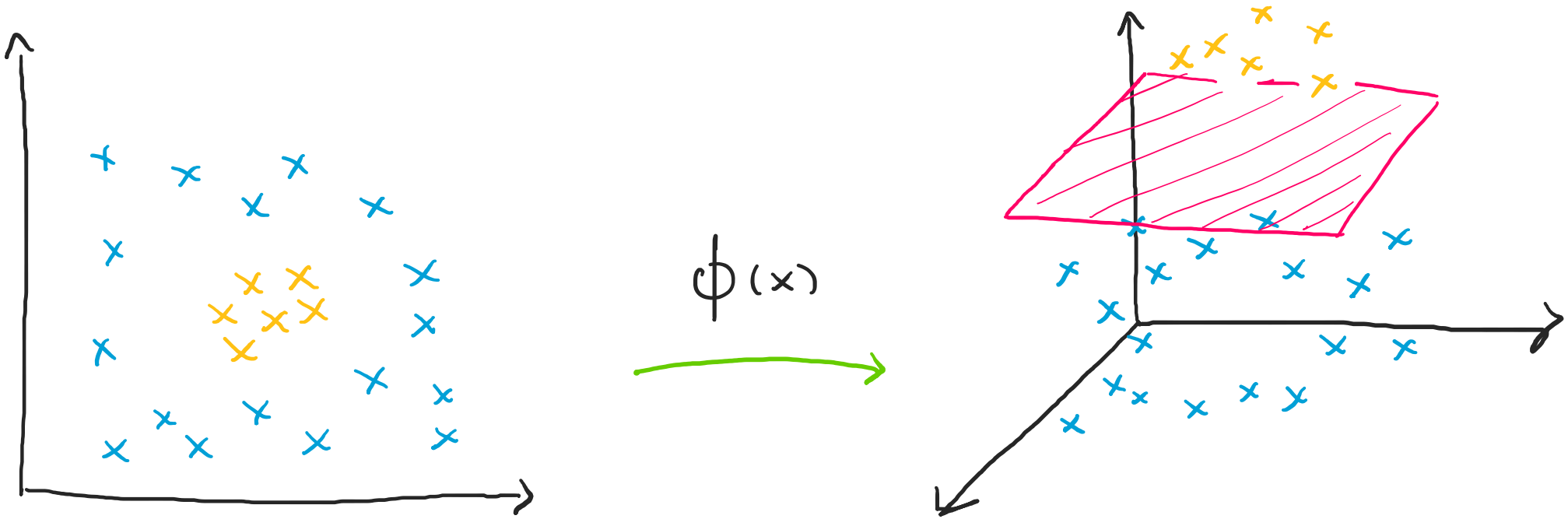
↳ Kernel ⟶ Feature Space

$\phi(x_i) \cdot \phi(s)$

$$\hat{y} = \text{label}(s) = \text{sign}\left(\sum \alpha_i y_i K(x_i, s) + b\right)$$

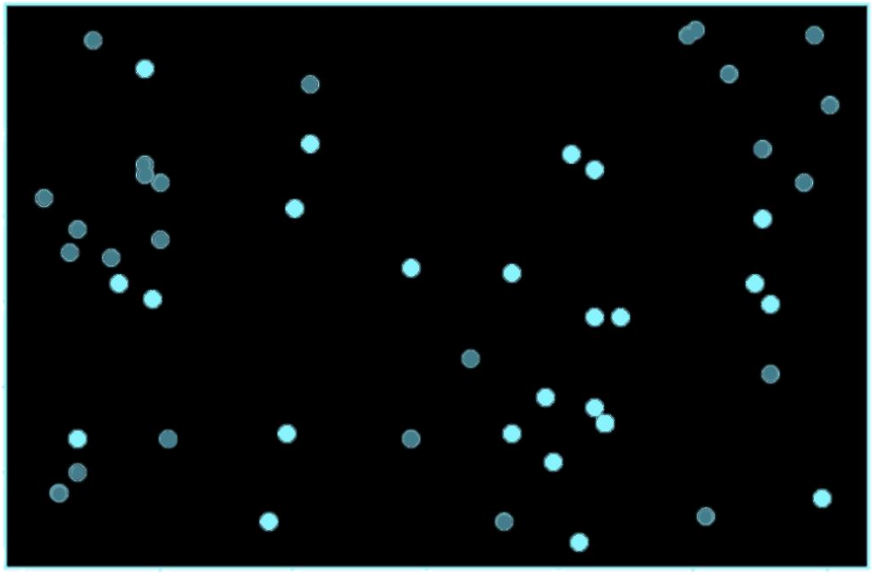
Dual Form

Kernel Methods: Hands on!

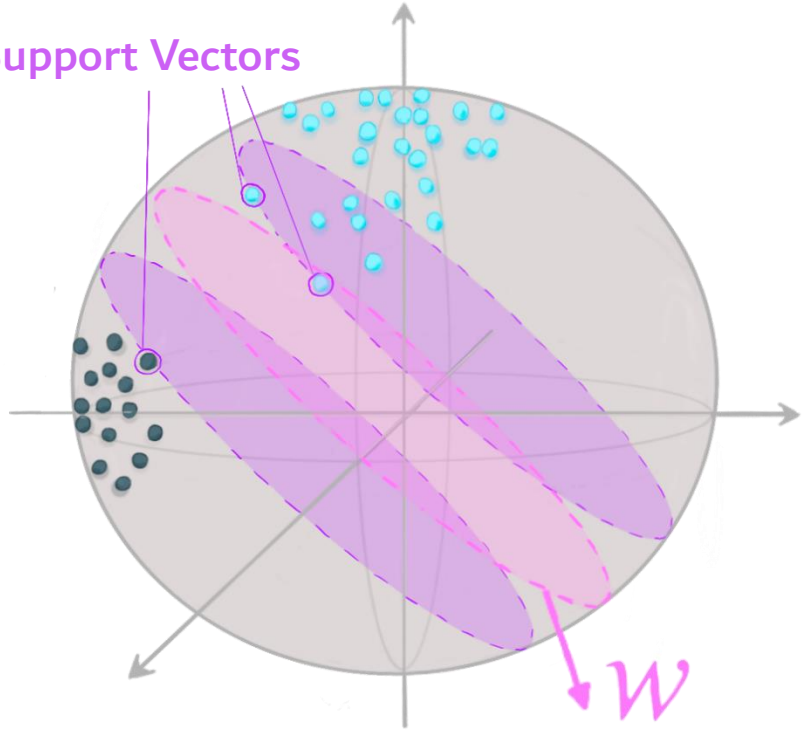


Quantum Feature Space

- **Non-linearly** separable datasets may become linearly separable by including new features.



- This transformation is called a **quantum feature map**



Quantum Kernel Estimation

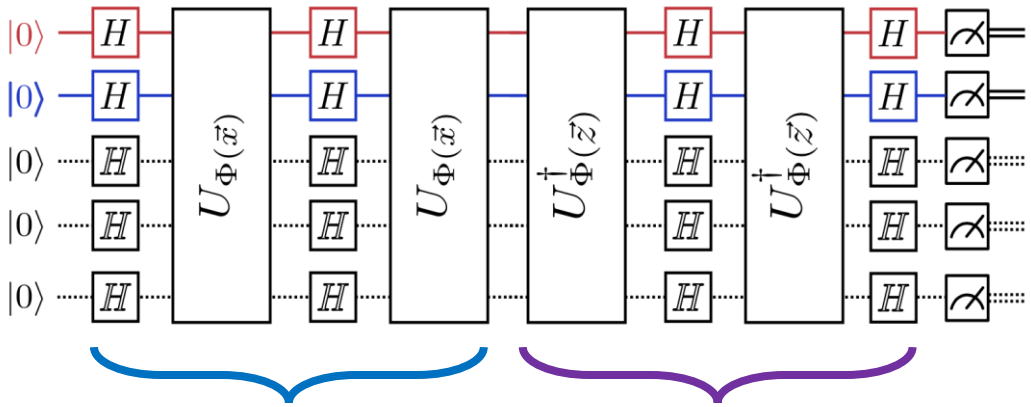
We use quantum computers to:

- encode the data
- estimate the kernel by measuring the **fidelity** between pairs of feature vectors

Classical computer feeded with the quantum kernel $\mathbf{K_{iz}}$ is then used to do the SVM according to:

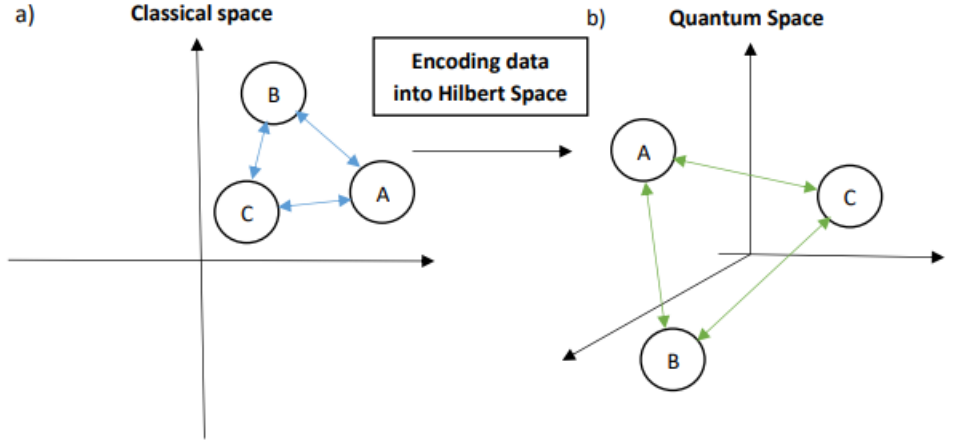
$$\hat{y} = \text{label}(z) = \text{sign}\left(\sum \alpha_i y_i K(x_i, z) + b\right)$$

$$|\langle \Phi(\vec{x}) | \Phi(\vec{z}) \rangle|^2 = |\langle 0^m | U^\dagger_{\Phi(\vec{x})} U_{\Phi(\vec{z})} | 0^m \rangle|^2$$



[1]

X Z



[2]

[1] V Havlicek et al, (2019), *Supervised learning with quantum-enhanced feature spaces,* Nature

[2] S Moradi et al, (2022), *Clinical data classification with NISQ computers,* Scientific reports

Classical Metrics

KERNEL POLARITY

- Finds the similarity between a kernel k with feature vectors $\{x_i\}$ and labels $\{y_i\}$ and its relative *ideal kernel matrix* $K^*_{ij} = y_i y_j$

$$P(k) = \sum_{i,j=1}^{n} y_i y_j k(x_i, x_j)$$

KERNEL-TARGET ALIGNMENT

- Normalized counterpart of the kernel polarity

$$TA(k) = \frac{P(k)}{||K^*||_F ||K||_F} \quad ; \quad ||K||_F = \sqrt{\sum_{i,j=1}^{n} k(x_i, x_j)^2} \quad ; \quad ||K^*||_F = \sqrt{\sum_{i,j=1}^{n} (y_i y_j)^2}$$

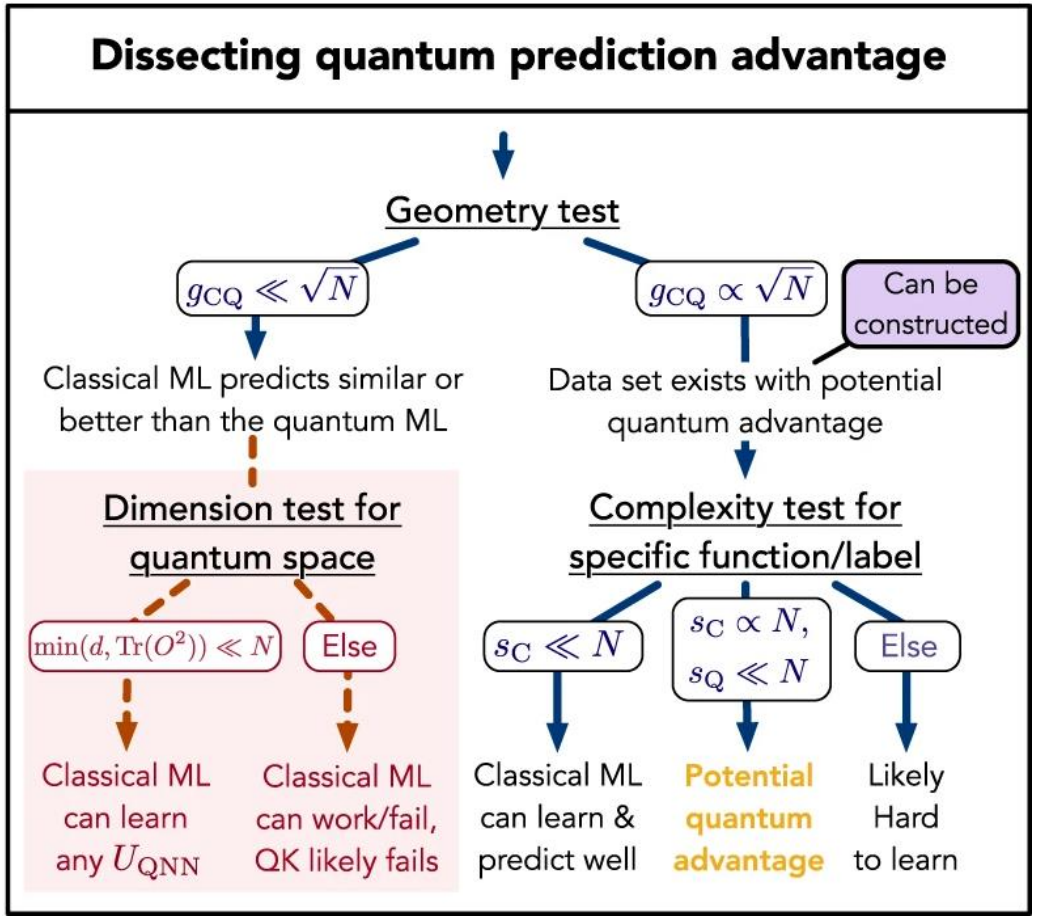
https://docs.pennylane.ai/en/stable/code/qml_kernels.html

Quantum Metrics

- Computations exponentially hard classically ✓
- Expressivity of QML hinder generalization because of overfitting ✗

So far, results found with trial and error but we need a reliable theoretical framework.

A priori methodology to assess quantum advantage according to data and kernels considered.



[3]

[3] HY Huang et al, (2021), *Power of Data in Quantum Machine Learning,* Nature Comm

Special Kernels

- **Structure learning kernel**
- **Projected kernel**

Both thought to control and design better feature maps

Goal →

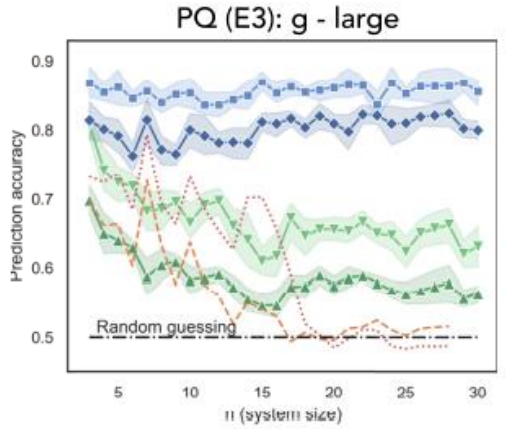
Effective quantum kernel hard-classically but with limited expressibility

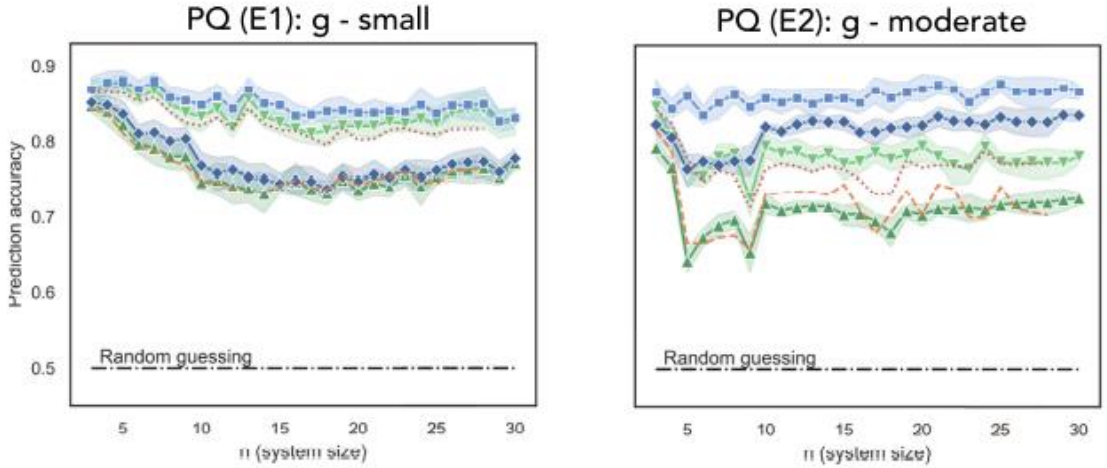
Projected Quantum Kernels

- Reduce the dimensionality of Feature Space by projection of QK:
  - ➢ To better generalize
  - ➢ To keep features into states classically hard

$$k^{|p}(x_i, x_j) = \sum_{k=1}^{m} \frac{Tr[\rho_k(x_i)\rho_k(x_j)]}{m}$$

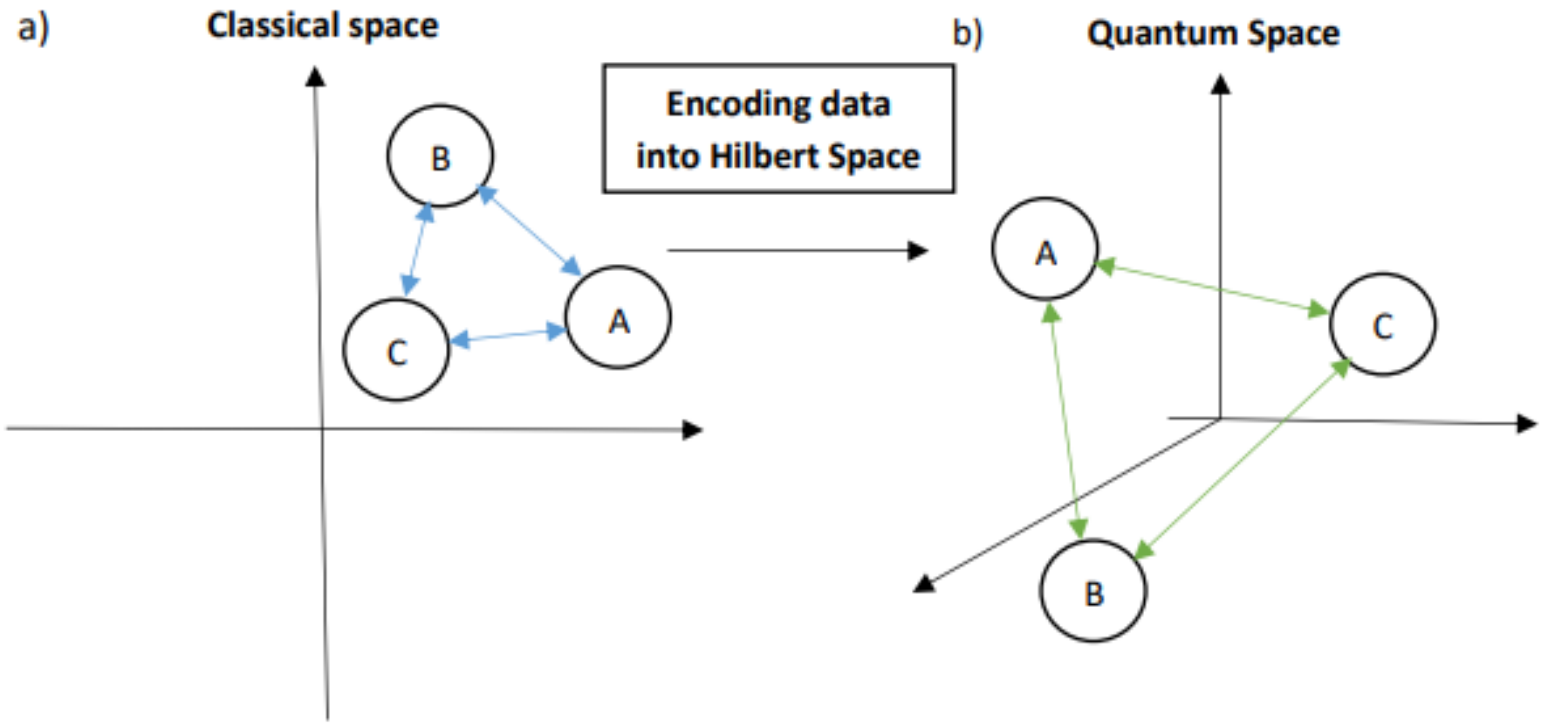
This projected kernel defines a feature map in a subspace of the large Hilbert Space. It can still express an high number of arbitrary functions (and their powers).



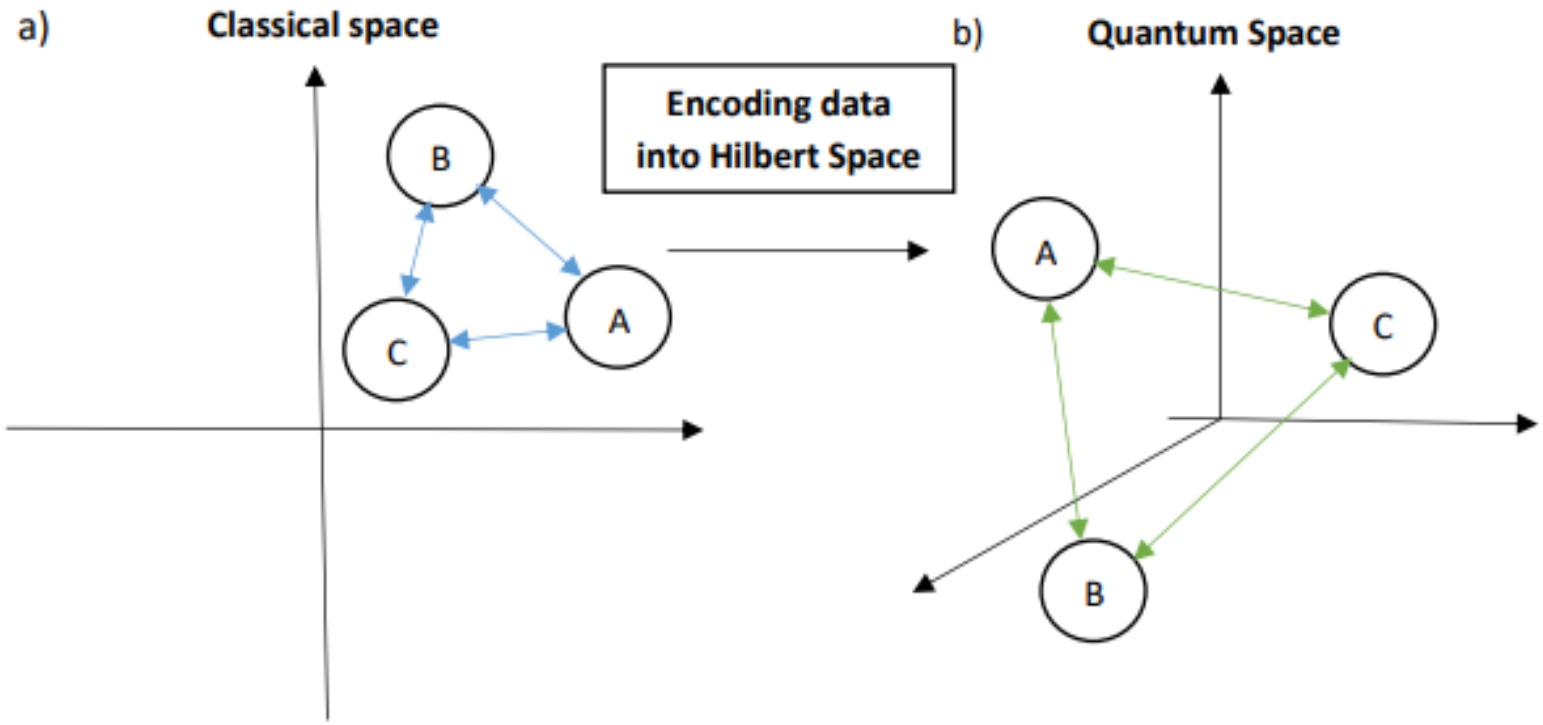


[3]

Quantum Kernels: Hands on!



Quantum Kernels: Hands on!



Bonus Material

- **Quantum Advantage Seeker for Kernels (Developed at CERN)**
- **Software Engineering Best Practices (Lived at CERN)**
- **Examples, Exercises and Tutorials (See notebook)**

Bonus Material: QuASK

Quantum Physics

[Submitted on 30 Jun 2022]

QuASK -- Quantum Advantage Seeker with Kernels

Francesco Di Marcantonio, Massimiliano Incudini, Davide Tezza, Michele Grossi

QuASK is a quantum machine learning software written in Python that supports researchers in designing, experimenting, and assessing different quantum and classical kernels performance. This software is package agnostic and can be integrated with all major quantum software packages (e.g. IBM Qiskit, Xanadu's Pennylane, Amazon Braket). QuASK guides the user through a simple preprocessing of input data, definition and calculation of quantum and classical kernels, either custom or pre-defined ones. From this evaluation the package provides an assessment about potential quantum advantage and prediction bounds on generalization error. Moreover, it allows for the generation of parametric quantum kernels that can be trained using gradient-descent-based optimization, grid search, or genetic algorithms. Projected quantum kernels, an effective solution to mitigate the curse of dimensionality induced by the exponential scaling dimension of large Hilbert spaces, are also calculated. QuASK can furthermore generate the observable values of a quantum model and use them to study the prediction capabilities of the quantum and classical kernels.

Bonus Material: Best Practices

Git

Keep track of code changes, making collaboration and code sharing easy.

Unit testing

Testing makes your code robust and gives you the confidence for doing refactorings.

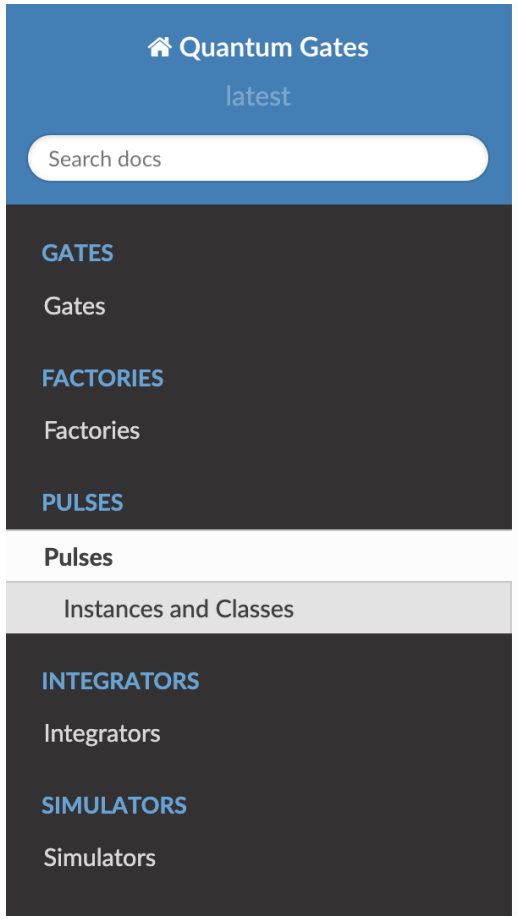
Packaging in Python

Package your code into a Python library, making the installation effortless.

Autodocs with Sphinx

Let the code document itself with tools like Sphinx, generating beautiful websites from code.

Bonus Material: Best Practices



/ Pulses

Edit on GitHub

Pulses

Gates on real quantum devices are commonly implemented as radiofrequency (RF) pulses. The present classes represent the pulse shapes, which can then be used to build to construct a gateset.

Instances and Classes

class **quantum_gates.pulses.Pulse**(*pulse: callable, parametrization: callable, perform_checks: bool = False, use_lookup: bool = False*) [source]

Bases: `object`

Parent class for pulses with basic utility.

Parameters:

- **pulse** (*callable*) – Function f: [0,1] -> R>=0: Waveform of the pulse, must integrate up to 1.
- **parametrization** (*callable*) – Function F: [0,1] -> [0,1]: Parameter integral of the pulse. Monotone with F(0) = 0 and F(1) = 1, as well as x <= y implies F(x) <= F(y).

Summary and Conclusion

1. Supervised classification problems can be complicated due to non linearity of the dataset
2. We choose as estimator a kernel
3. Based on the problem under study we can use Quantum Kernel methods
4. We learned how to implement them using the largest quantum platform available, i.e. Qiskit
5. We compare with classical methods with metrics
6. QML is advancing and we face new problems!

THANK YOU FOR YOUR ATTENTION

F. Di Marcantonio
UPV/EHU Quantum Center
QTI CERN

roman.wixinger@gmail.com
francesco.di.marcantonio@cern.ch

R. Wixinger
ETH Zurich

01.08.2023

Backup Slides

WHAT A MESS!

THINGS TO KEEP IN MIND WHEN WORKING WITH QUANTUM KERNELS

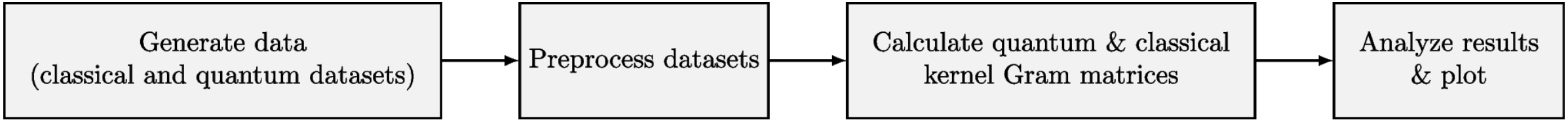
- Does my task well adapt to the use of quantum kernels?
- Does my technique work with a specific dataset?
- Can I prove it?

- Do I need expressibility reduction techniques?
- Are projected kernels ok? Do I need something different?

- Is my data clean? Do I need preprocessing?

- Is my code working?

PIPELINE


Generate data
(classical and quantum datasets)
Preprocess datasets
Calculate quantum & classical
kernel Gram matrices
Analyze results
& plot

Can we automatize this pipeline?

DATA GENERATION

```
> python3.9 -m quask get-dataset
The available datasets are:
  0: iris                          - classification  - classical
  1: Fashion-MNIST                 - classification  - classical
  2: liver-disorders               - regression      - classical
  3: delta_elevators               - regression      - classical
  4: Q_Fashion-MNIST_2_E1          - regression      - quantum
  5: Q_Fashion-MNIST_4_E2          - regression      - quantum
  6: Q_Fashion-MNIST_8_E3          - regression      - quantum
Which dataset will you generate?: 0
Where is the output folder?: .
Saved file ./iris_X.npy
Saved file ./iris_y.npy
```

PREPROCESSING

```
> python3.9 -m quask preprocess-dataset
Choose a random seed: 2323
Choose a random seed for splitting training and testing set: 3232
Where is the output folder: .
Specify the path of classical feature data X (.npy array): iris_X.npy
        Loaded X having shape (150, 4) (150 samples of 4 features)
Specify the path of classical feature data T (.npy array): iris_y.npy
        Loaded Y having shape (150,) (150 labels)
        Labels have type int64 thus this is a classification problem
        The dataset has 3 labels being [0, 1, 2] distributed as it follows:
                Class 0 is present 50 times (33.33 %)
                Class 1 is present 50 times (33.33 %)
                Class 2 is present 50 times (33.33 %)
Do you want to pick just the first two classes? [y/N]: y
Do you want to undersample the largest class? [y/N]: n
Do you want to oversample the smallest class? [y/N]: n
Do you want to apply preprocessing to the features? [y/N]: y
Do you want to apply PCA (numerical data only)? [y/N]: y
How many components you want?: 2
Do you want to apply FAMD (both numerical and categorical data)? [y/N]: n
Do you want to scale each field from 0 to 1 (MinMaxScaler)? [y/N]: y
Which percentage of data must be in the test set?: 0.5
Saved file ./X_train.npy
Saved file ./y_train.npy
Saved file ./X_test.npy
Saved file ./y_test.npy
```

DEFINITION & USE OF QUANTUM KERNELS

```
> python3.9 -m apply-kernel
Where is X_train (npy file)?: X_train.npy
Where is y_train (npy file)?: y_train.npy
Where is X_test (npy file)?: X_test.npy
Where is y_test (npy file)?: y_test.npy
Do you want to apply a fixed kernel [Y] or a trainable one [N]? [y/N]: y
The available fixed kernels are:
  0: linear_kernel
  1: rbf_kernel
  2: poly_kernel
  3: zz_quantum_kernel
  4: projected_zz_quantum_kernel
  5: random_quantum_kernel
  6: projected_random_quantum_kernel
Which kernel Gram matrix will you generate?: 0
Where is the output folder?: .
Saved file ./training_linear_kernel.npy
Saved file ./testing_linear_kernel.npy
```

DEFINITION & USE OF QUANTUM KERNELS

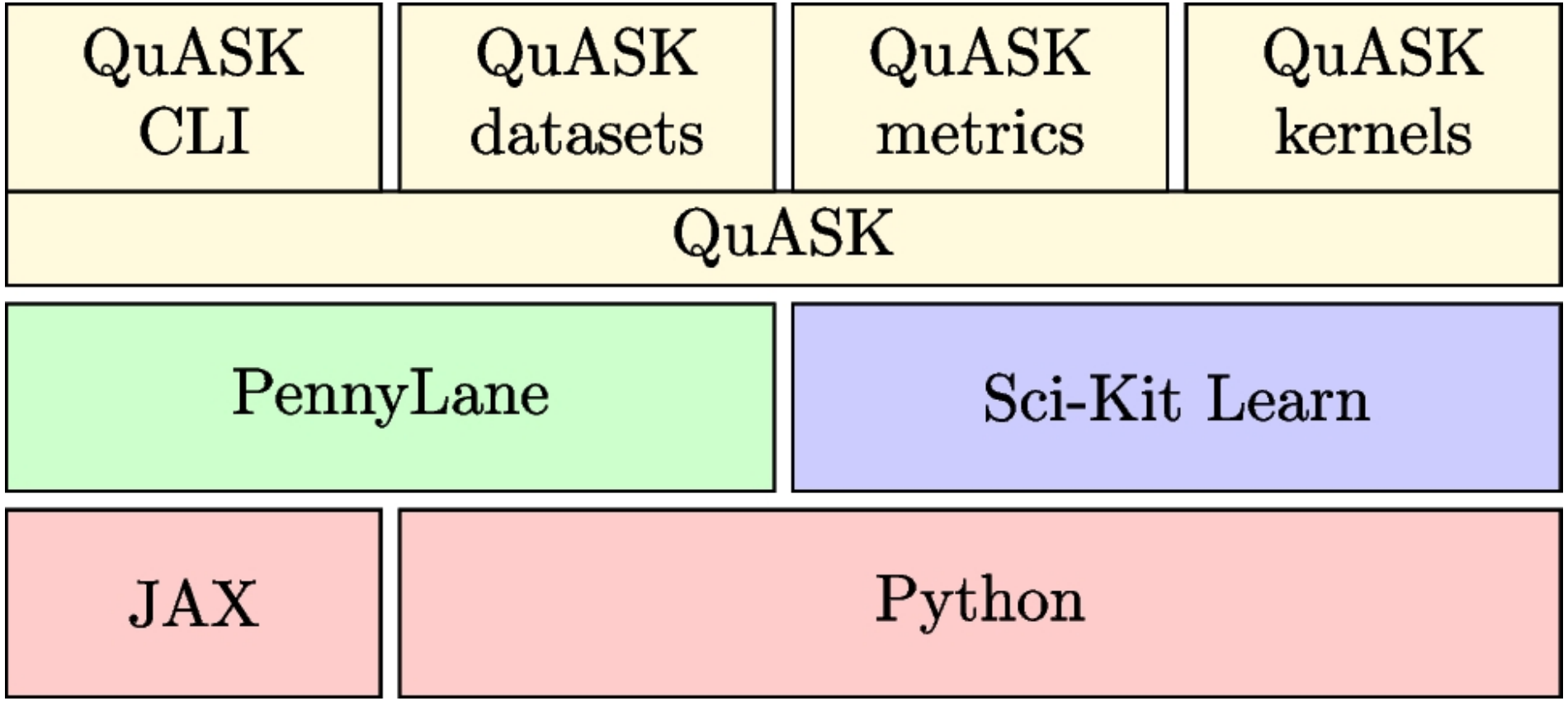
```
> python3.9 -m apply-kernel
Where is X_train (npy file)?: X_train.npy
Where is y_train (npy file)?: y_train.npy
Where is X_test (npy file)?: X_test.npy
Where is y_test (npy file)?: y_test.npy
Do you want to apply a fixed kernel [Y] or a trainable one [N]? [y/N]: n
Choose an embedding for your data (rx, ry, rz, zz): rx
Choose an embedding for your data (hardware_efficient, tfim, ltfim, zz_rx): tfim
Choose a number of layers: 1
Choose an optimizer (adam, grid): adam
Choose a reward metric to maximize (kernel-target-alignment, accuracy, geometric-differen
    ce, model-complexity): accuracy
Choose a random seed: 434343
WARNING:absl:No GPU/TPU found, falling back to CPU. (Set TF_CPP_MIN_LOG_LEVEL=0 and rerun
    for more info.)
.....................................................................................
Where is the output folder?: .
Where is the output folder?: .
Saved file ./training_trainable_rx_tfim_1_adam_accuracy.npy
Saved file ./testing_trainable_rx_tfim_1_adam_accuracy.npy
```

DATA ANALYSIS

```
python3.9 -m quask plot-metric \
--metric accuracy \
--train-gram training_linear_kernel.npy --train-y Y_train.npy \
--test-gram testing_linear_kernel.npy --test-y Y_test.npy \
--label linear \
--train-gram training_rbf_01_kernel.npy --train-y Y_train.npy \
--test-gram testing_rbf_01_kernel.npy --test-y Y_test.npy \
--label rbf \
--train-gram training_rbf_001_kernel.npy --train-y Y_train.npy \
--test-gram testing_rbf_001_kernel.npy --test-y Y_test.npy \
--label rbf \
--train-gram training_poly_kernel.npy --train-y Y_train.npy \
--test-gram testing_poly_kernel.npy --test-y Y_test.npy \
--label poly \
--train-gram training_trainable_rx_tfim_1_adam_accuracy.npy \
--train-y Y_train.npy \
--test-gram testing_trainable_rx_tfim_1_adam_accuracy.npy --test-y  Y_test.npy \
--label 'trainable quantum kernel'
```

STRUCTURE

SOFTWARE STACK



CLI VS API

- QuASK can be integrated within your application

```python
from quask.datasets import load_who_life_expectancy_dataset

dataset = load_who_life_expectancy_dataset()
X = dataset['X']
y = dataset['y']
# use the dataset for any purpose
```

```python
from quask.metrics import calculate_approximate_dimension

kernel_gram_matrix = ...
d = calculate_approximate_dimension(kernel_gram_matrix)
```

```python
from quask.kernels import projected_zz_quantum_kernel

X = ... # feature matrix
kernel_gram_matrix = projected_zz_quantum_kernel(X, X, 0.1) # gamma = 0.1
```

CLI VS API

CLI

- Zero code
  (no need to know Python,
  no need to debug)
- Shallow learning curve
- Perfect for fast prototyping

API

- Maximum flexibility
- Easily integrable with existing software
- Extends QuASK with new functionalities