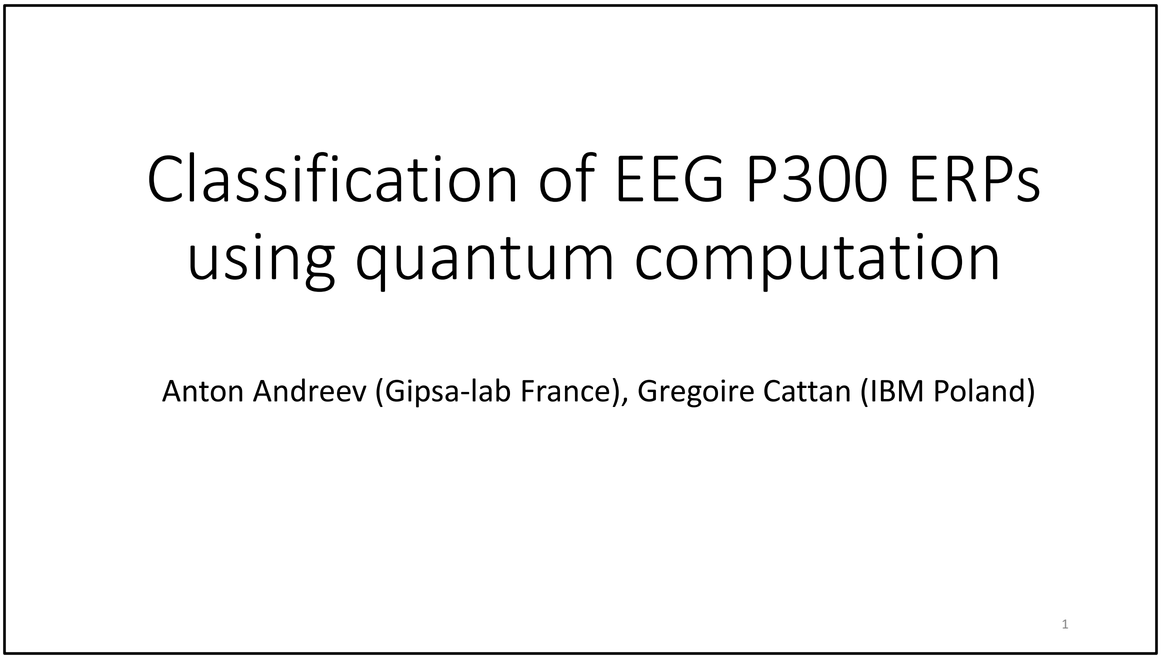
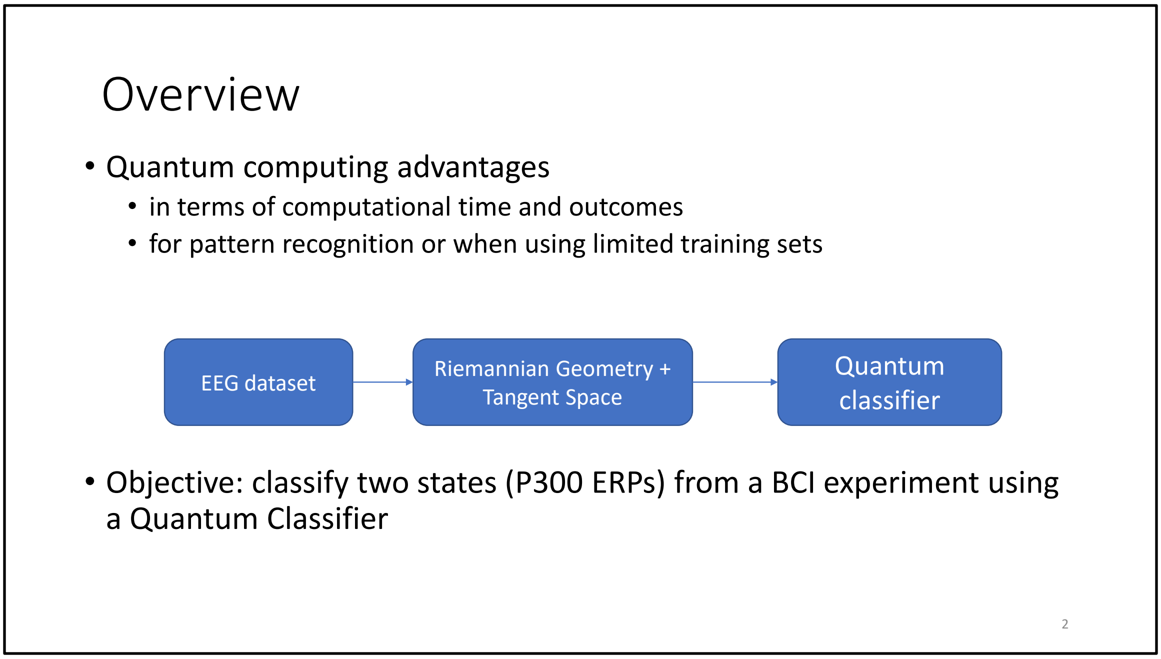


Classification of EEG P300 ERPs using quantum computation

Anton Andreev (Gipsa-lab France), Gregoire Cattan (IBM Poland)



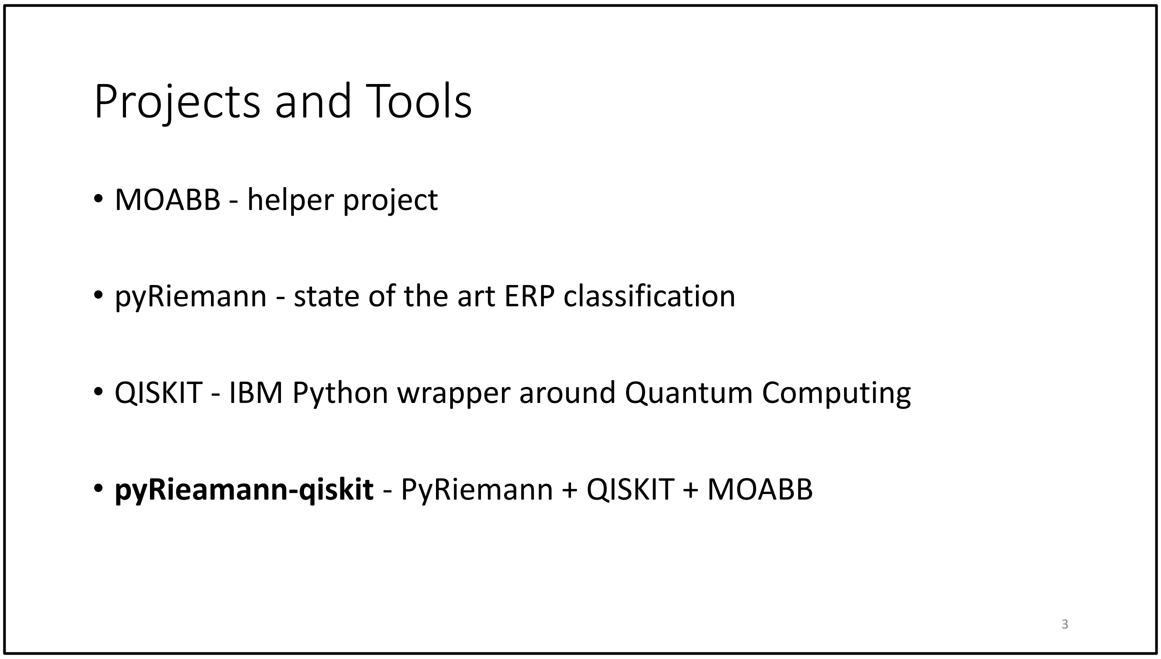
Overview

- Quantum computing advantages
  - in terms of computational time and outcomes
  - for pattern recognition or when using limited training sets

EEG dataset → Riemannian Geometry + Tangent Space → Quantum classifier

- Objective: classify two states (P300 ERPs) from a BCI experiment using a Quantum Classifier

Literature on quantum computing suggests it may offer an advantage as compared with classical computing in terms of computational time and outcomes, such as for pattern recognition or when using limited training sets.

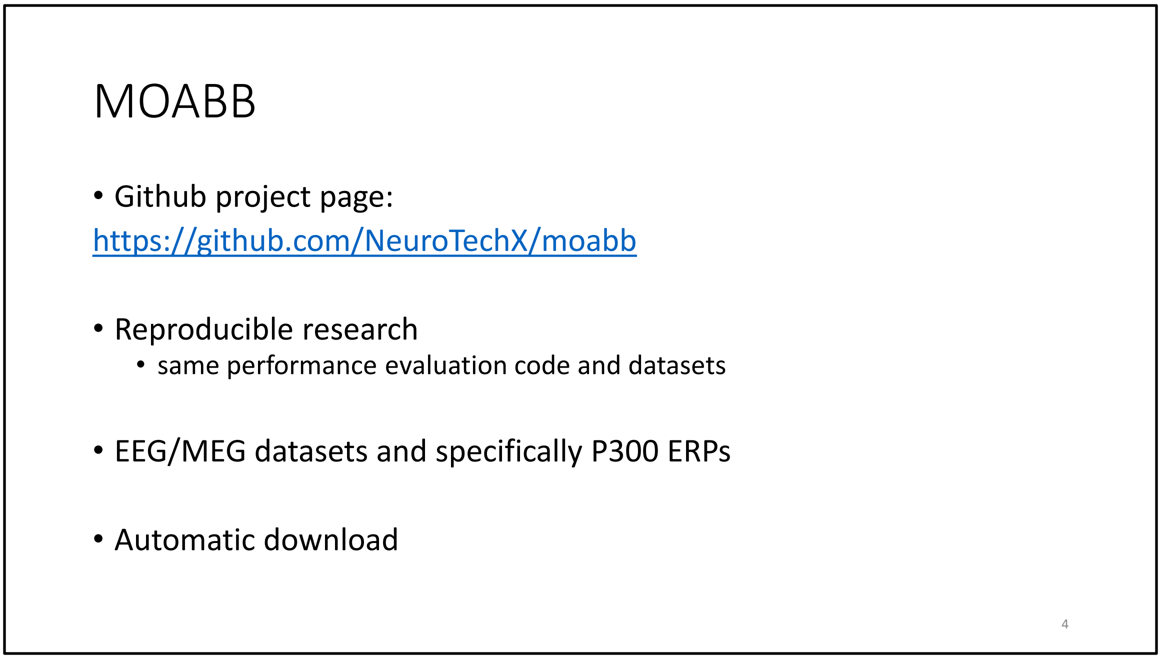


Projects and Tools

- MOABB - helper project
- pyRiemann - state of the art ERP classification
- QISKIT - IBM Python wrapper around Quantum Computing
- **pyRieamann-qiskit** - PyRiemann + QISKIT + MOABB

3

pyRieamann-qiskit combines the 3 projects above,so that we can easily apply Quantum computing on EP EEG signals.



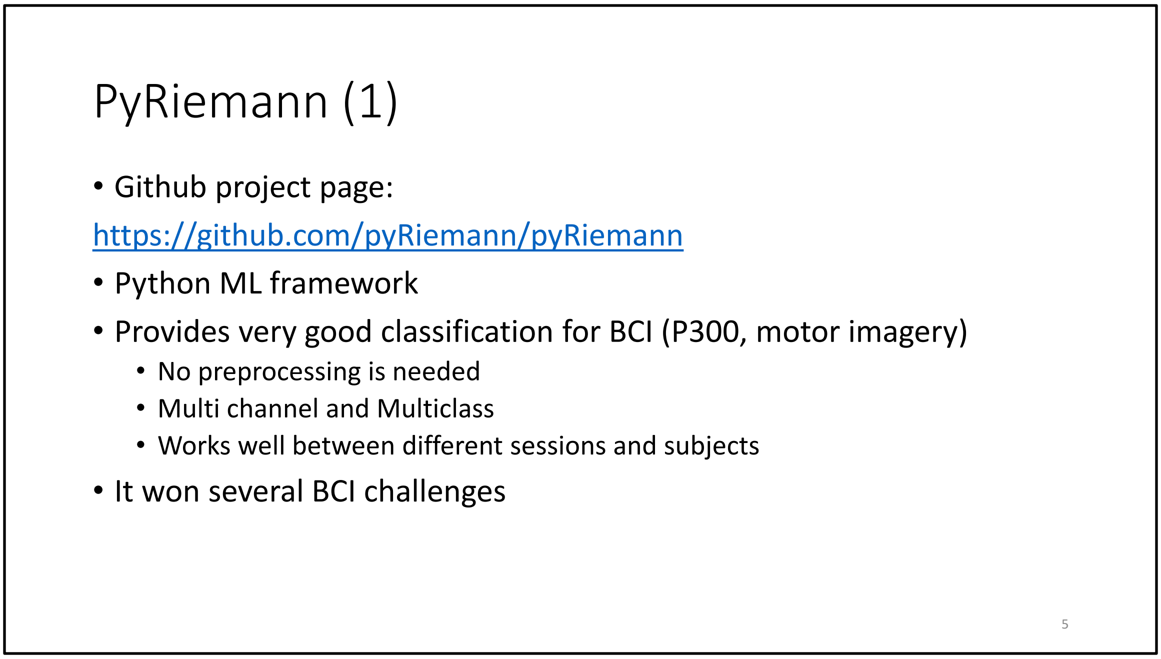
MOABB

- Github project page:
https://github.com/NeuroTechX/moabb

- Reproducible research
  - same performance evaluation code and datasets

- EEG/MEG datasets and specifically P300 ERPs

- Automatic download



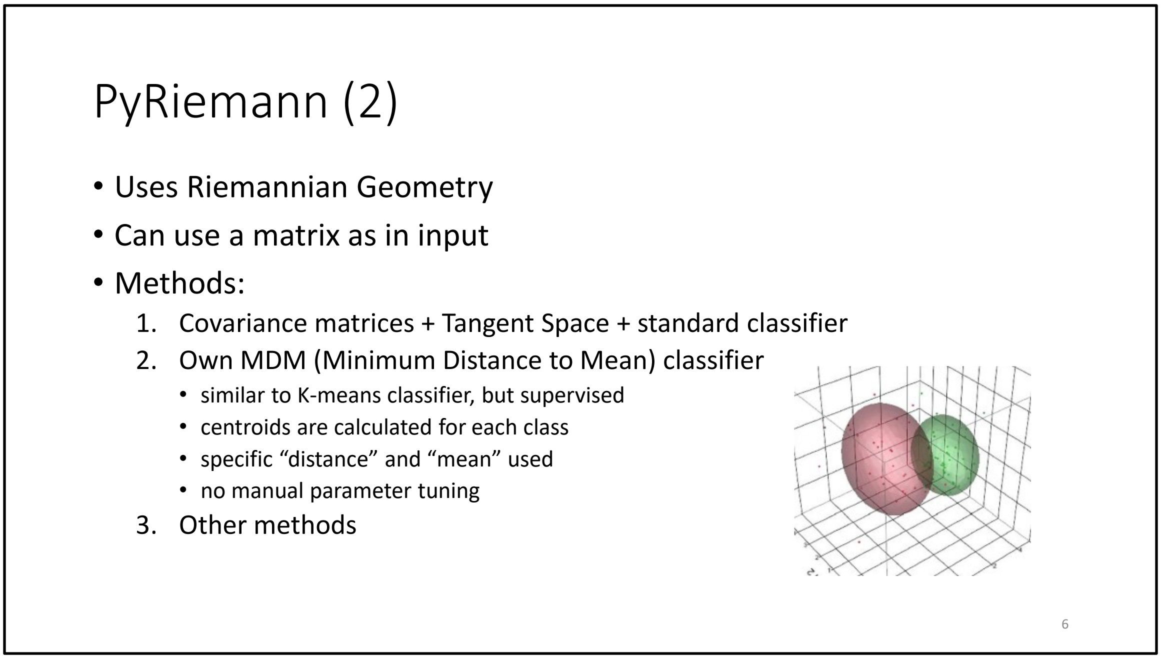
PyRiemann (1)

- Github project page:

https://github.com/pyRiemann/pyRiemann

- Python ML framework
- Provides very good classification for BCI (P300, motor imagery)
  - No preprocessing is needed
  - Multi channel and Multiclass
  - Works well between different sessions and subjects
- It won several BCI challenges

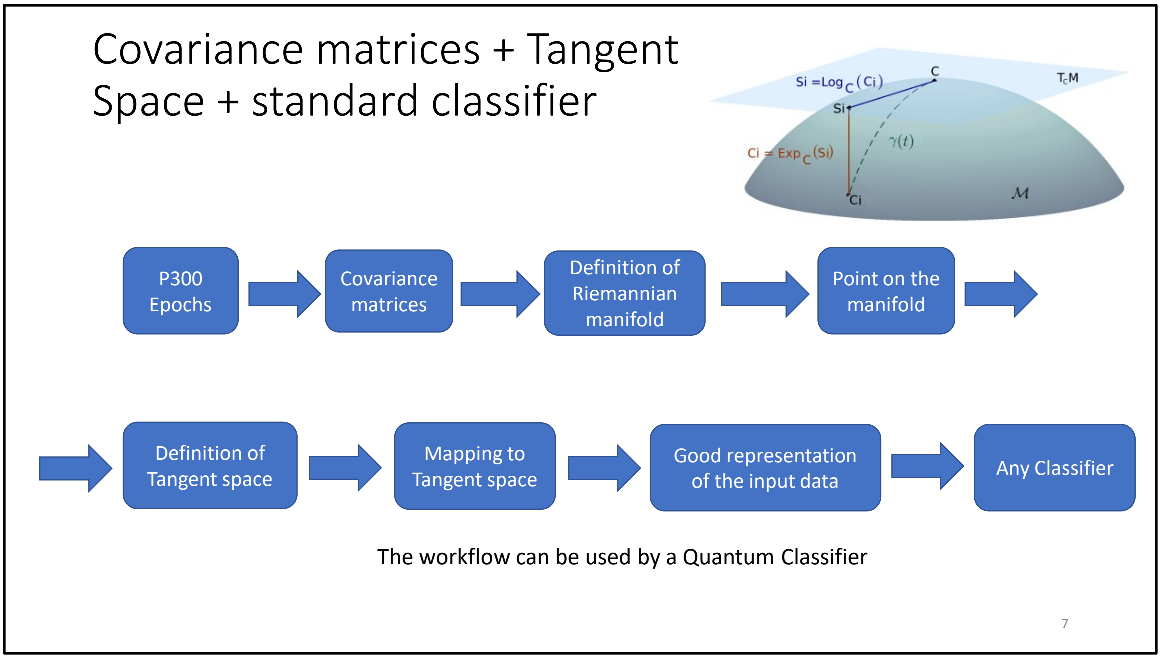
PyRiemann provides state of the art classification of ERP signals.



PyRiemann (2)

- Uses Riemannian Geometry
- Can use a matrix as in input
- Methods:
  1. Covariance matrices + Tangent Space + standard classifier
  2. Own MDM (Minimum Distance to Mean) classifier
     - similar to K-means classifier, but supervised
     - centroids are calculated for each class
     - specific “distance” and “mean” used
     - no manual parameter tuning
  3. Other methods

K-means is unsupervised in general, but here in MDM we use labeled data to build the two clusters for the classes: with or without P300 ERP.
We need to properly define the *distance* and *mean* we are going to use. “distance” as defined using Riemannian Geometry.
Inherited from the *geometric* distance, the geometric mean is an appropriate descriptor of the central tendency (expected value) for the variance, while, inherited from the Euclidean distance, the arithmetic mean is not.



Covariance matrices + Tangent Space + standard classifier

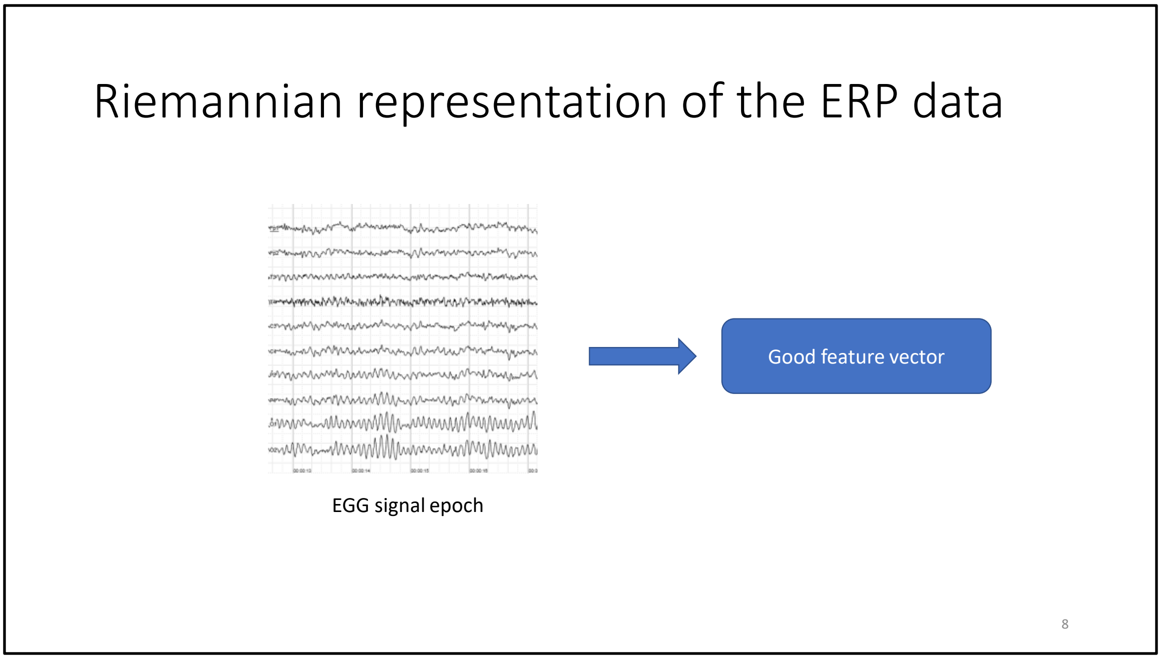
P300 Epochs → Covariance matrices → Definition of Riemannian manifold → Point on the manifold → Definition of Tangent space → Mapping to Tangent space → Good representation of the input data → Any Classifier

The workflow can be used by a Quantum Classifier

7

- From each P300 epoch we calculate a “covariance matrix”. We want to preserve the spatial information in the covariance matrix. Usually the covariance matrix is vectorized (all rows of the matrix are put one after another) to form a single input vector to be used by any classifier, but here this is avoided. The spatial covariance matrix Cp is calculated using the single trail Xp (channels x samples): Cp = 1 / (T-1) * Xp * Xp(transposed).
- Cp is symmetric by definition
- With sufficient data Cp becomes positive definitive, otherwise it must be regularized (there are known algorithms for that)
- Next the space of SPDs creates a manifold (based on the training data)
- This manifold becomes Riemannian manifold M by adopting an Affine Invariant Metric
- Now we can apply Riemannian Geometry for M
- Each Cp can be represented in M
- For each point on the manifold M a tangent space T can be defined
- We need to select a *reference point C* as to where to construct the tangent space T. Usually this *reference point is* the *Geometric Mean.* The *Geometric Mean* might not be the best *reference point*, but usually is a very good choice. The Geometric mean is calculated iteratively

- Each point on the manifold (each covariance matrix) can be projected on T using log mapping (and back using exponential mapping). A kind of optimization is C to become the identity matrix and to use parallel transport (parallel translation)
- So the Riemannian distance in M is well approximated by Euclidean distance in T – so T (which is 2 dimensional) acts as a very good approxomation of M
- Any classifier can be used in T



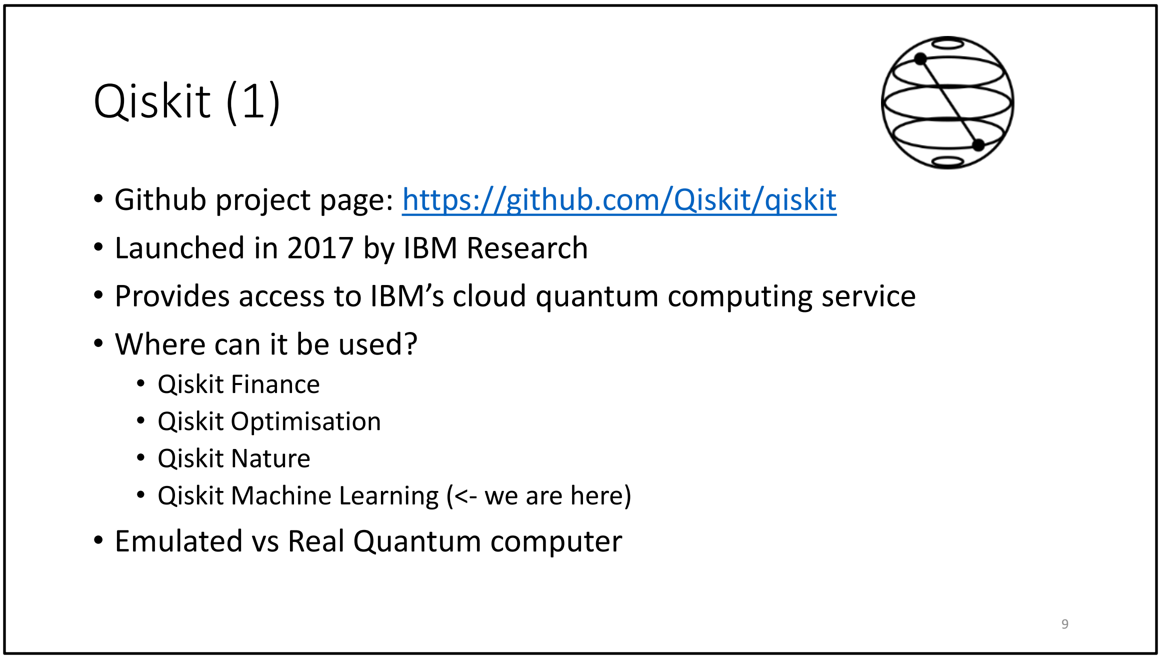
Riemannian representation of the ERP data

EGG signal epoch

Good feature vector

8

By using PyRiemann we get a good a feature vector that can be used for Classical or Quantum Classification.



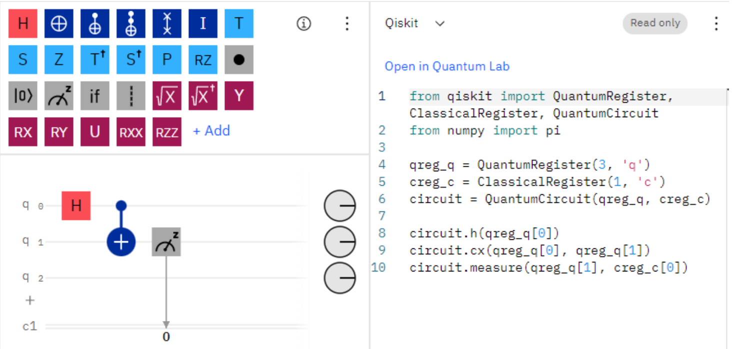
Qiskit (1)

- Github project page: https://github.com/Qiskit/qiskit
- Launched in 2017 by IBM Research
- Provides access to IBM’s cloud quantum computing service
- Where can it be used?
  - Qiskit Finance
  - Qiskit Optimisation
  - Qiskit Nature
  - Qiskit Machine Learning (<- we are here)
- Emulated vs Real Quantum computer

We are in the exploration phase of Quantum computers. But the fact that anyone can access a real quantum computer is impressive and will help progress the domain.
Qiskit framework can be used on two levels: at a low level for Quantum models (as shown on the next slide) or at a higher level where a Quantum classifier is already provided (and we can focus on the hyper parameters).
Qiskit has 3 ML algorithms: QSVM (Quantum-enhanced Support Vector Machine) and VQC (Variational Quantum Classifier) and QGAN (Quantum Generative Adversarial Network).

Qiskit (2)

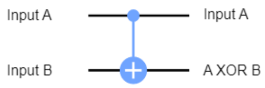
- Quantum circuit
- Quantum bits
- Operations
- Registers:
  - Quantum (q1,q2)
  - Classical (c1)
- Code generated:
  - Python
  - QASM
- Result is the measurement of q1 available c1



```
from qiskit import QuantumRegister,
ClassicalRegister, QuantumCircuit
from numpy import pi

qreg_q = QuantumRegister(3, 'q')
creg_c = ClassicalRegister(1, 'c')
circuit = QuantumCircuit(qreg_q, creg_c)

circuit.h(qreg_q[0])
circuit.cx(qreg_q[0], qreg_q[1])
circuit.measure(qreg_q[1], creg_c[0])
```



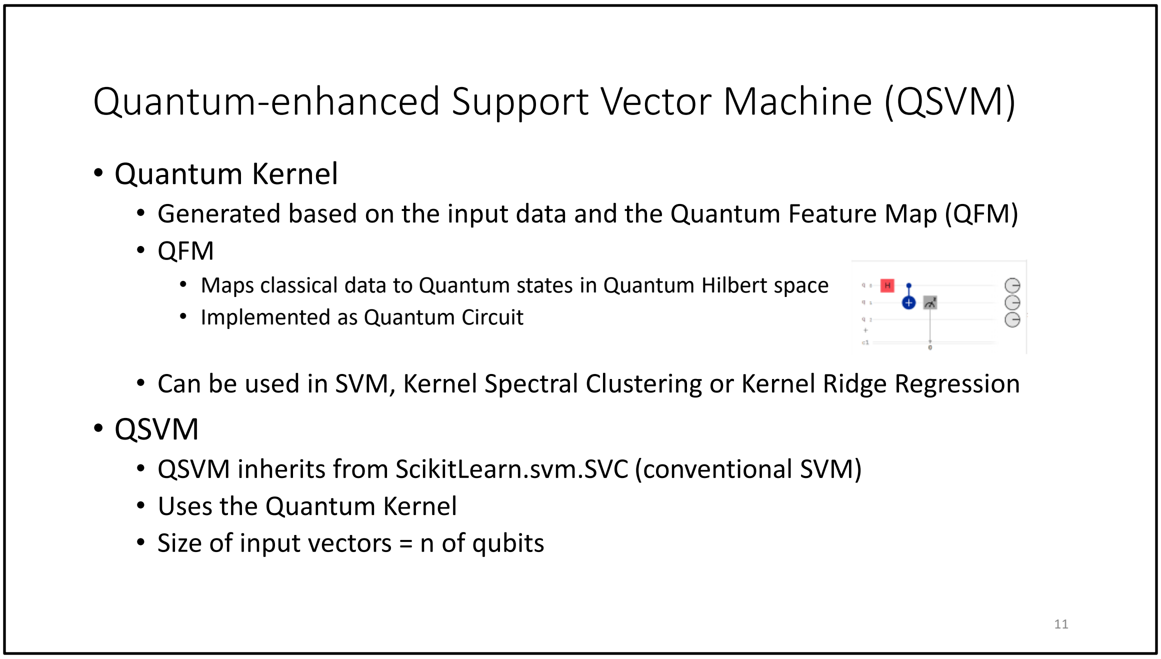
https://quantum-computing.ibm.com

With Qiskit we are doing Quantum Programming.
Qiskit compiles an algorithm to a quantum circuit.
Quantum bits can be manipulated in ways that can only be described by quantum physics.
Each quantum bit can seen as a “quantum register”. Classical bits form “classical register”.
Quantum circuit is a combination of operations on these registers. Here the quantum circuit is comprised from the quantum bits q0, q1, q2 and a normal bit c1.
Quantum operations include quantum gates, such as the Hadamard gate, as well as operations that are not quantum gates, such as the measurement operation.
From the Web interface you can generate either Qiskit python code or QASM (Open Quantum Assembly Language).
Qubits are always initialized to give the output 0.

Quantum operations:
https://qiskit.org/textbook/ch-states/atoms-computation.html
H – is a Hadamard gate
+ - is *a CX (CNOT) gate on control qubit 0 and target qubit 1 (a classical* XOR gate). It looks at its two input bits to see whether they are the same or different. Next, it

overwrites the target qubit with the answer. The target becomes 0 if they are the same, and 1 if they are different. Another way of explaining the CNOT is to say that it does a NOT on the target if the control (the first quantum bit) is 1, and does nothing otherwise.
X – is a NOT gate (circle with + inside, exactly after Hadamard gate)

Textbook on Qiskit: https://qiskit.org/textbook/preface.html



Quantum-enhanced Support Vector Machine (QSVM)

- Quantum Kernel
  - Generated based on the input data and the Quantum Feature Map (QFM)
  - QFM
    - Maps classical data to Quantum states in Quantum Hilbert space
    - Implemented as Quantum Circuit
  - Can be used in SVM, Kernel Spectral Clustering or Kernel Ridge Regression
- QSVM
  - QSVM inherits from ScikitLearn.svm.SVC (conventional SVM)
  - Uses the Quantum Kernel
  - Size of input vectors = n of qubits

11

Going to the higher dimensional space called “Feature Space” is done using a “kernel”. The job of the kernel is to transform linearly inseparable data to linearly separable space.
A Quantum Feature Map $\boldsymbol{V}(\Phi(\vec{x}))$ converts classical data to quantum data. It maps to quantum Hilbert space with specific property of the dot product. The reason of choosing a QFM is to get the quantum advantage.
Based on the QFM we construct a quantum kernel (during the training phase).
The QFM is modeled using a quantum circuit and quantum operations as shown in the previous slide.
A different QFM will produce a different kernel. The default Quantum Feature Map (QFM) is ZZFeatureMap:
https://qiskit.org/documentation/stubs/qiskit.circuit.library.ZZFeatureMap.html

Links and extras:
The Quantum Kernel python class in Qiskit is:
qiskit_machine_learning.kernels.quantum_kernel
Check also on IBM web-site: qiskit-tutorials/qiskit-machine-learning/03_quantum_kernel.ipynb
QSVC code: https://github.com/Qiskit/qiskit-machine-

learning/blob/1f01764186ad5e6fd4b88deb4dfec1e5cbb05d03/qiskit_machine_learning/algorithms/classifiers/qsvc.py

Article Quantum Feature Map: https://shubham-agnihotri.medium.com/quantum-machine-learning-102-qsvm-using-qiskit-731956231a54

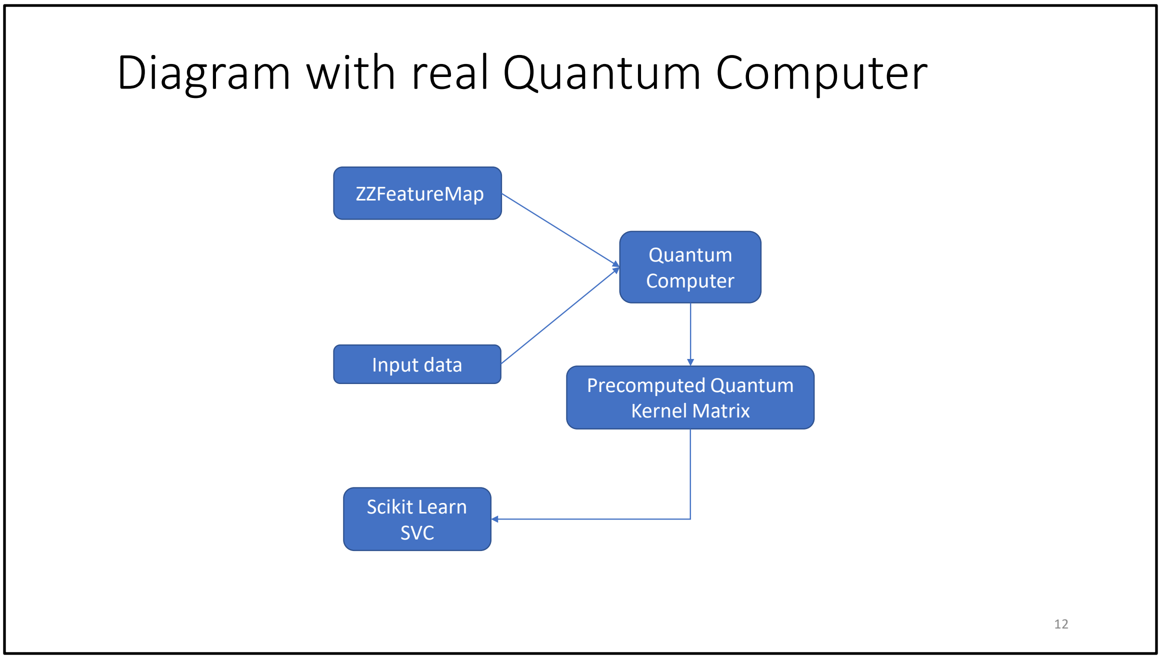


Diagram with real Quantum Computer

ZZFeatureMap

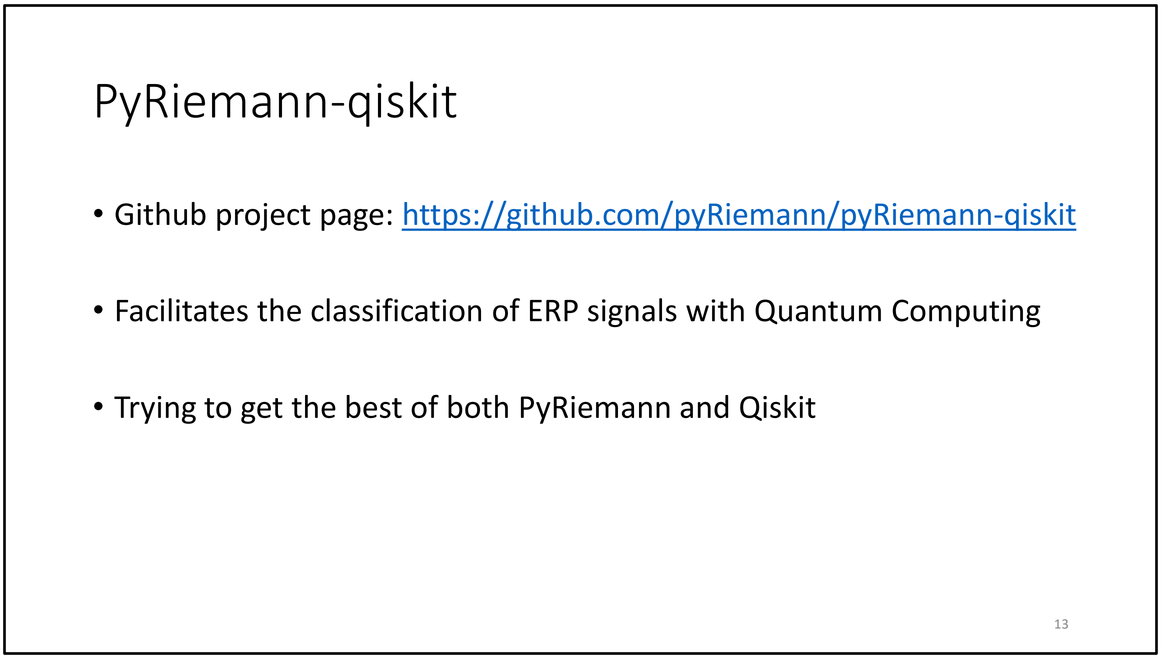
Input data

Quantum Computer

Precomputed Quantum Kernel Matrix

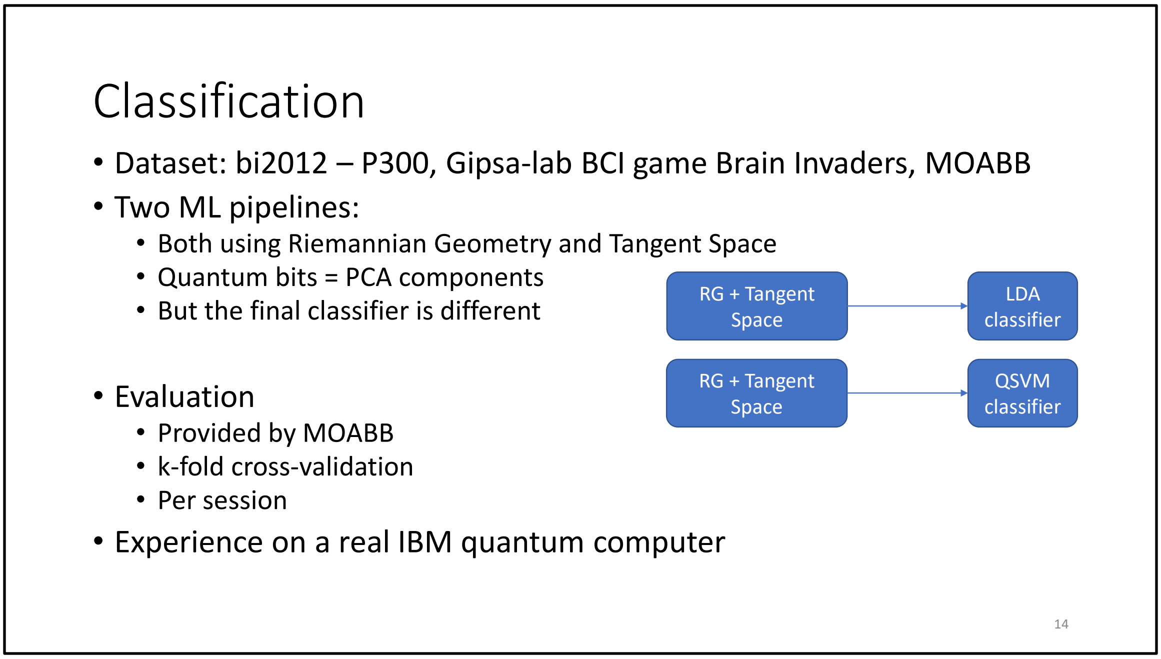
Scikit Learn SVC

The algorithm starts on your laptop, then the model representing the kernel and the data are sent to a real quantum computer. The result is a kernel that is compatible with Scikitlearn’s implementation of Support Vector Machines.



PyRiemann-qiskit

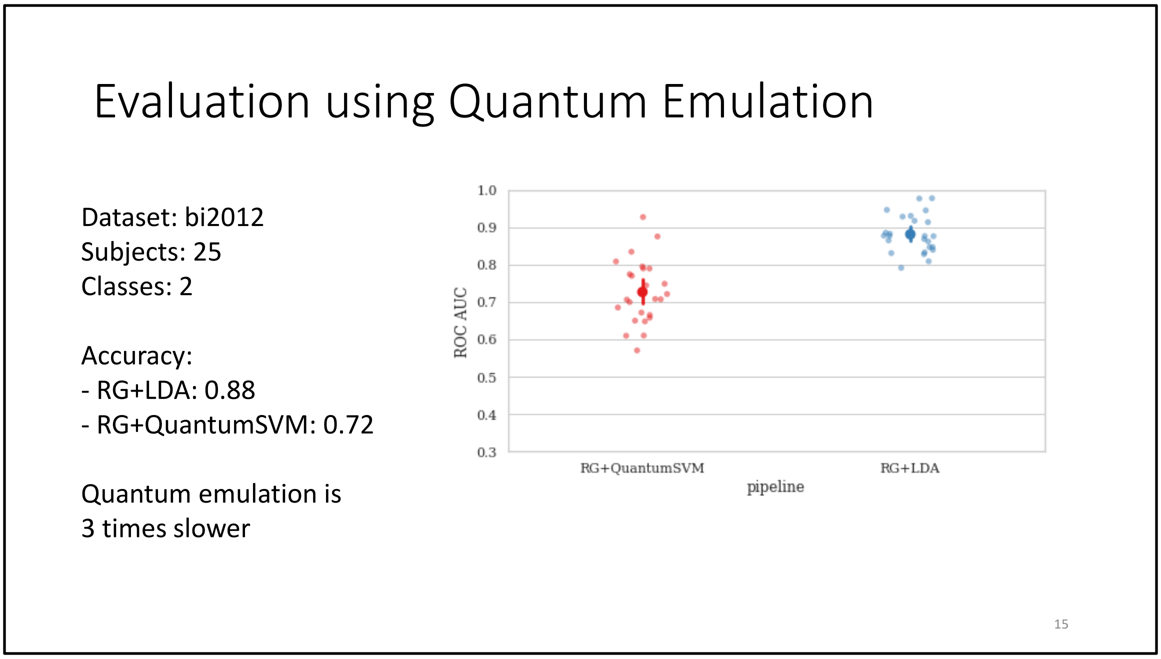
- Github project page: https://github.com/pyRiemann/pyRiemann-qiskit
- Facilitates the classification of ERP signals with Quantum Computing
- Trying to get the best of both PyRiemann and Qiskit



Classification

- Dataset: bi2012 – P300, Gipsa-lab BCI game Brain Invaders, MOABB
- Two ML pipelines:
  - Both using Riemannian Geometry and Tangent Space
  - Quantum bits = PCA components
  - But the final classifier is different
- Evaluation
  - Provided by MOABB
  - k-fold cross-validation
  - Per session
- Experience on a real IBM quantum computer

The following pipeline “Xdawn spatial filter, Riemannian Geometry, Tangent Space, PCA” is used to generate the same feature vectors before applying a simple LDA or QSVM.
One subject can have several recorded sessions playing the BCI game Brain Invaders. “Within-session” evaluation uses k-fold cross-validation to determine train and test sets for each separate session for each subject
http://moabb.neurotechx.com/docs/generated/moabb.evaluations.WithinSessionEvaluation.html



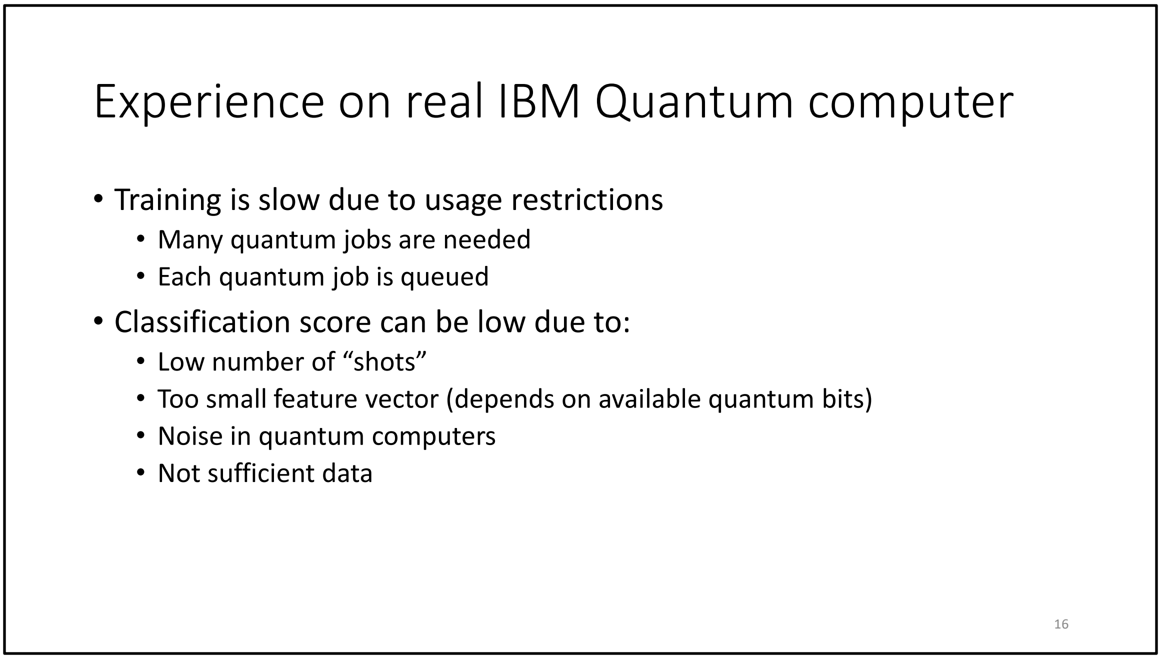
Evaluation using Quantum Emulation

Dataset: bi2012
Subjects: 25
Classes: 2

Accuracy:
- RG+LDA: 0.88
- RG+QuantumSVM: 0.72

Quantum emulation is
3 times slower

Evaluation of the two pipelines.
Both finish in 5 minutes on a regular laptop.

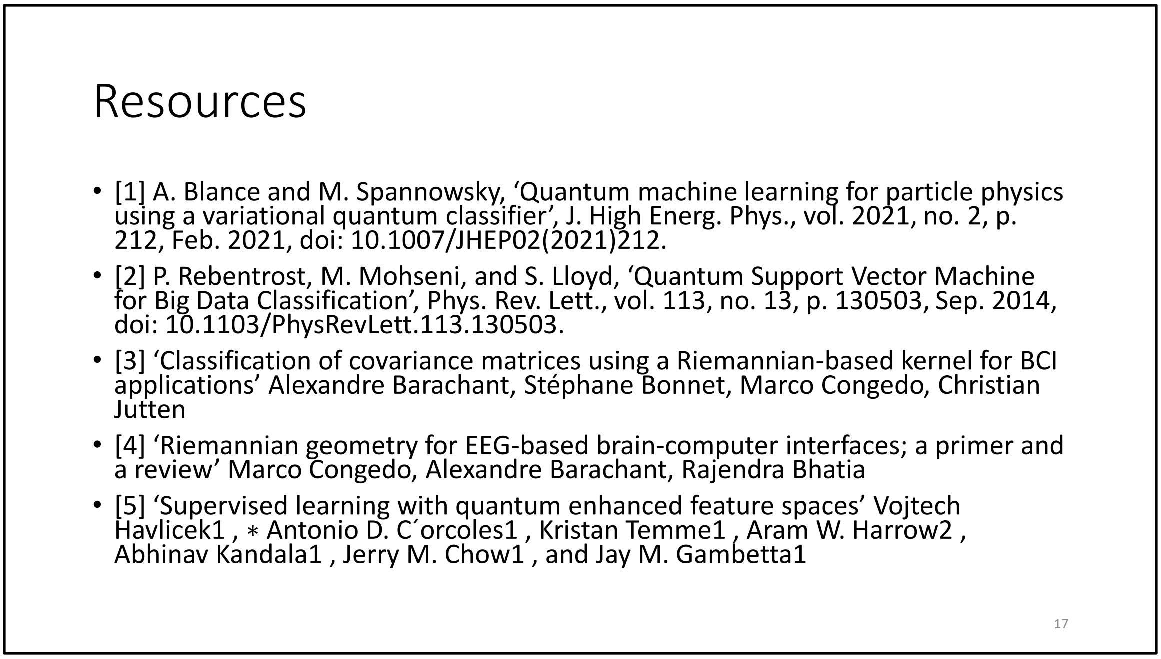


Experience on real IBM Quantum computer

- Training is slow due to usage restrictions
  - Many quantum jobs are needed
  - Each quantum job is queued
- Classification score can be low due to:
  - Low number of “shots”
  - Too small feature vector (depends on available quantum bits)
  - Noise in quantum computers
  - Not sufficient data

16

Training is slow because many users are “fighting” for the same “quantum computer”.
Building the Quantum Feature Map is done by performing many “Quantum jobs” (a unit of work executed on a Quantum computer).
“Shots” – the number of times the circuit is executed. More shots gives more confidence in the result, but it also takes much longer to process.
The size of the generated feature vector (based on RG + Tangent Space) is limited to the number of available quantum bits.
A real quantum computer should perform less accurately than an emulated one.
Although an emulated quantum computer can include emulation of quantum noise.
Adding more data (P300 ERP epochs) means more quantum jobs and more time to complete. But limiting the input data however means less training of the QSVM.



Resources

- [1] A. Blance and M. Spannowsky, ‘Quantum machine learning for particle physics using a variational quantum classifier’, J. High Energ. Phys., vol. 2021, no. 2, p. 212, Feb. 2021, doi: 10.1007/JHEP02(2021)212.
- [2] P. Rebentrost, M. Mohseni, and S. Lloyd, ‘Quantum Support Vector Machine for Big Data Classification’, Phys. Rev. Lett., vol. 113, no. 13, p. 130503, Sep. 2014, doi: 10.1103/PhysRevLett.113.130503.
- [3] ‘Classification of covariance matrices using a Riemannian-based kernel for BCI applications’ Alexandre Barachant, Stéphane Bonnet, Marco Congedo, Christian Jutten
- [4] ‘Riemannian geometry for EEG-based brain-computer interfaces; a primer and a review’ Marco Congedo, Alexandre Barachant, Rajendra Bhatia
- [5] ‘Supervised learning with quantum enhanced feature spaces’ Vojtech Havlicek1 , ∗ Antonio D. C´orcoles1 , Kristan Temme1 , Aram W. Harrow2 , Abhinav Kandala1 , Jerry M. Chow1 , and Jay M. Gambetta1

[1,2] Advantages of Quantum Computing

[3.4] Classification with the state of art method using Riemannian geometry

[5] Quantum Support Vector Classifier

Questions?