Variational Quantum Classifier for Heart Disease Prediction

Team members

- Șofariu Adrian[1]
- Șoptelea Sebastian[1]
- Lung Rodica Ioana[1]
- Mahu Gheorghe[2]
- Mihoc Tudor-Dan[1]

1. Centre for the Study of Complexity, Babeş Bolyai University, Cluj Napoca
2. Centrul de Inovare şi Transfer tehnologic MAVIS, UMF Iaşi

Problem

Explore the implementation of a framework to predict heart disease using a Variational Quantum Classifier (VQC).

Problem

Binary classification of patient data to predict the presence of heart disease

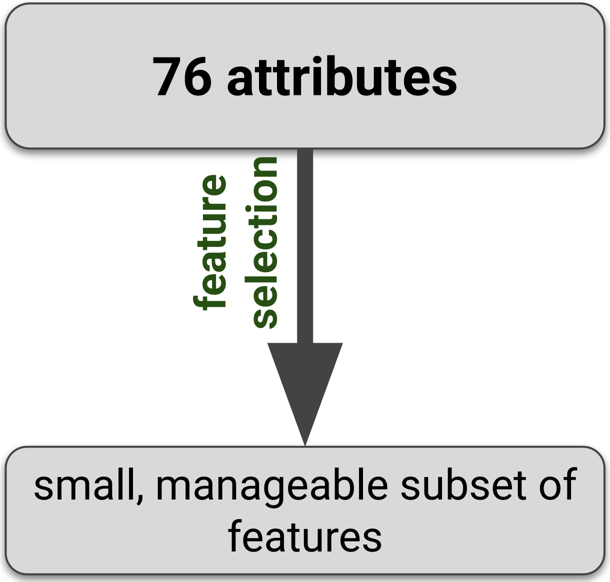
Variational Quantum Classifier (VQC)

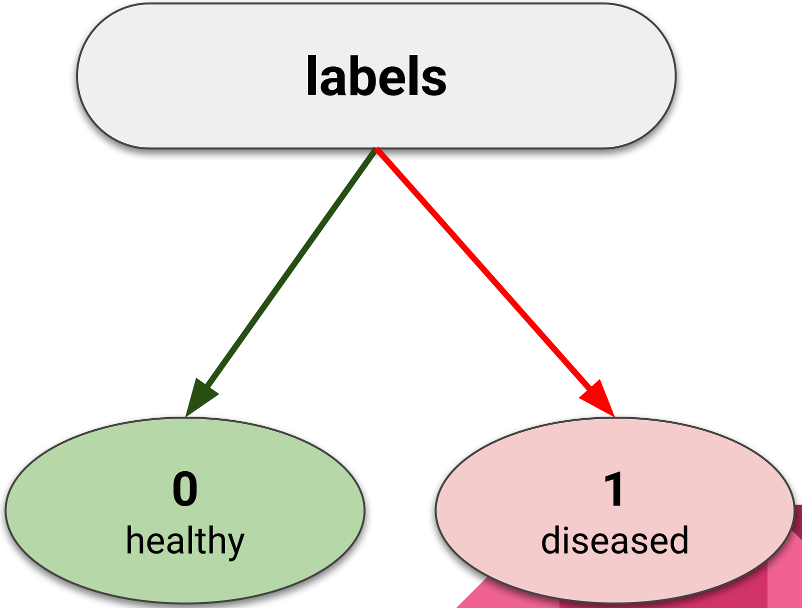
Evolutionary computation ➡ Quantum Circuit Parameter Optimization

Pipeline

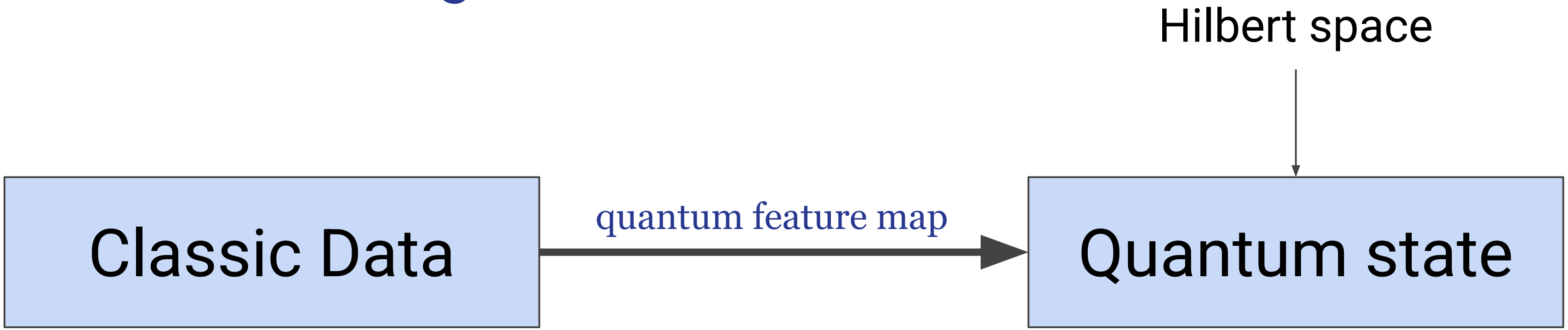
- Prerequisites Setup
- Defined Constants
- Data Preparation
- Quantum Circuit Preparation
- Utility Function
- Classification and Cost Evaluation
- Execution and Optimization
- Result Visualization

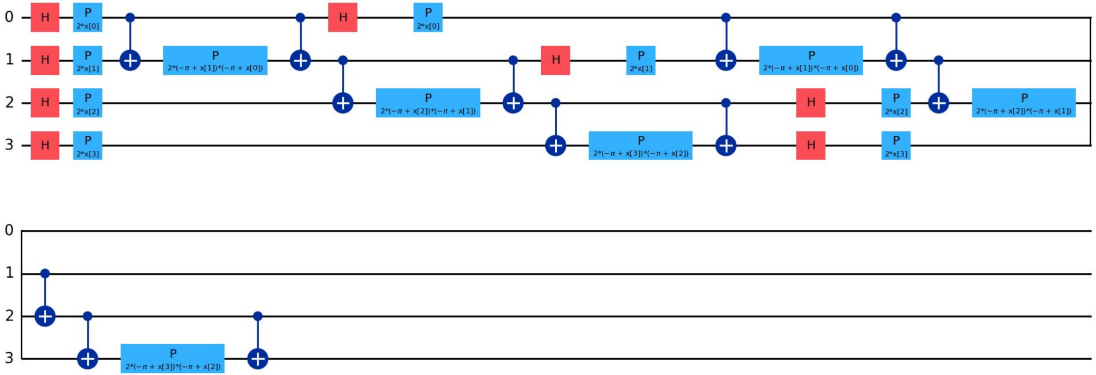
Database - UCI Heart Disease Dataset



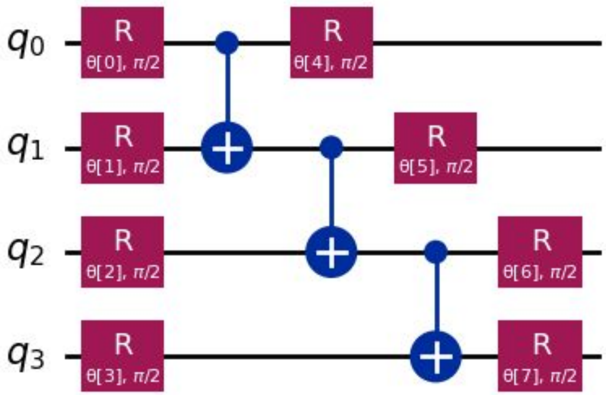


Data embedding





Quantum Circuit Preparation




```
ansatz = real_amplitudes(
    num_qubits = QUBIT_COUNT,  # Number of qubits = number of features
    reps = 1,                  # One repetition layer of rotations + entanglement
    entanglement = 'linear'    # Linear entanglement between qubits
)
```

Quantum Circuit Composition ➡ data-encoding feature map + the parameterized ansatz circuit

Quantum Circuit Utility Functions

- **Parameter Dictionary Creation for Quantum Circuit**
- **Label Assignment Based on Bit String Parity**
- **Probability Calculation from Quantum Measurement Counts**

Classification and Cost Evaluation

Processing Details

- For each input feature vector:
  - Binds feature data and ansatz parameters to the quantum circuit.
  - Adds measurement operations to read out qubit states.
  - Transpiles/prepares the circuit for execution.
- Runs all circuits in batch on the quantum sampler.
- Converts measurement results into class probabilities using parity-based labeling.

Output

Returns a list of dictionaries, each containing class labels and their associated probabilities for the corresponding input feature vector.

```
def classify(x_list: list, params: list, sampler: any, pm: any) -> list:
    qc_list = [ ]
    for x in x_list:
        # Bind input features and ansatz parameters to the circuit
        classifier = circuit.assign_parameters(get_data_dict(params, x))
        # Add measurement operations on all qubits
        classifier.measure_all()
        # Transpile or prepare the circuit for the backend
        transpiled_circuit = pm.run(classifier)
        qc_list.append(transpiled_circuit)
    # Execute all circuits on the simulator or quantum backend
    results = sampler.run(qc_list, shots=SHOT_COUNT).result()
    probs = []
    for qc in results:
        # Extract counts of measurement outcomes
        counts = qc.data.meas.get_counts()
        # Convert counts to probabilities for each class label
        prob = return_probabilities(counts)
        probs.append(prob)
    return probs
```

Cost Functions

Binary Cross-Entropy Cost

Calculates the *Binary Cross-Entropy* (BCE) loss for a predicted probability corresponding to the expected class label.

Cost Function for Quantum Circuit Training

Calculates the *average loss over the dataset* by comparing predicted probabilities against expected labels using Binary Cross-Entropy.

We will define an **Objective Function** – a cost function that evaluates how well, for a given parameter set, the training data is classified.

Quantum Circuit Parameter Optimization

- **CMA-ES** (Covariance Matrix Adaptation Evolution Strategy):
  - Evolutionary algorithm suitable for noisy or non-convex problems
  - Handles parameter noise using cma.NoiseHandler
  - Controlled by CMA_ITER_COUNT
- **COBYLA** (Constrained Optimization BY Linear Approximations):
  - Gradient-free optimizer based on linear approximations
  - Ideal for small parameter spaces and quick iterations
  - Controlled by COBYLA_ITER_COUNT

Quantum Circuit Parameter Optimization

Processing Details

1. **Initialize Parameters** Randomly initialized in the range [0, 2π] to match typical rotation gate requirements for angle encoding.
2. **Define Objective Function** The cost function evaluates how well a given parameter set classifies training data.
3. **Run Optimization** Depending on the flag, either CMA-ES or COBYLA minimizes the cost over several iterations.

Results and Validation

accuracy in simulations without noise

Cobyla	CMA-ES	SVM non-quantum method
86%	75%	81.81%

accuracy in simulations with noise

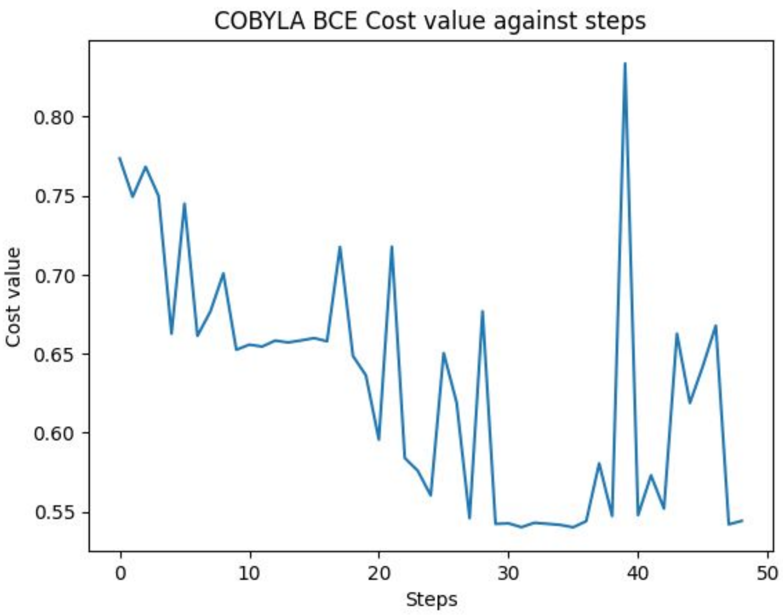
Cobyla	CMA-ES	SVM non-quantum method
70% – 85% more frequent towards 85%	74% when using a NoiseHandler, closer to ~83%	81.81%

Running on QPU

- Training Iterations:
  - Completed 49 out of 70 planned iterations before the sessions were continuously marked as “Pending” on the IBM Quantum platform.
- Backends & Region:
  - Used ibm_fez in the US-East region.
  - Also attempted to use ibm_kingston (least busy at the time), but sessions were pending and did not execute.
- Runtime & Queueing:
  - Total runtime: ~7–8 hours.
  - Each iteration took approximately 2m 20s to 2m 50s of actual QPU time.
  - Additional time lost due to queue delays — queues ranged from 20 to 35 users.

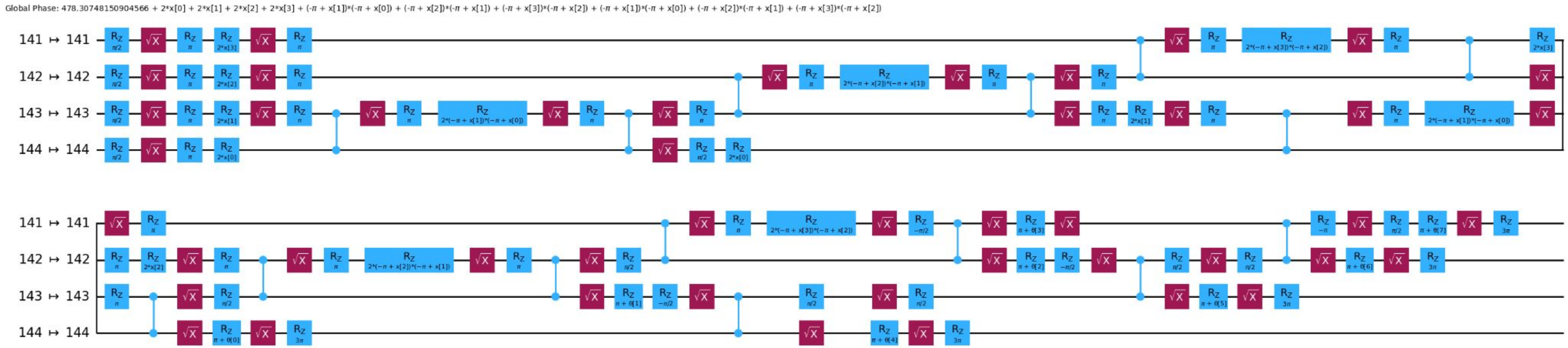
Running on QPU

- **Training Performance:**
  - Binary Cross Entropy (BCE) cost plateaued around 0.54 and did not drop lower during training.
- **Accuracy Evaluation:**
  - Achieved an accuracy of ~74%.
  - Standard deviation = 0.0000, with a 95% confidence interval of [0.7386, 0.7386].
  - The lack of variation may be due to the use of a fixed random seed (seed = 42) during testing.
- **Comparison with Local Simulations:**
  - Accuracy on local (noisy) simulators varied widely: 70%–85%, depending on the run.
  - While the QPU run scored slightly below some simulated runs, it is still promising, especially given that the cost function typically stabilizes around iteration 65–70 — beyond the 49 reached here.




```
Running model testing over 30 iterations...

Iteration 1/30... Test Accuracy - 0.7386 (65/88 correct predictions)
Iteration 2/30... Test Accuracy - 0.7386 (65/88 correct predictions)
Iteration 3/30... Test Accuracy - 0.7386 (65/88 correct predictions)
Iteration 4/30... Test Accuracy - 0.7386 (65/88 correct predictions)
Iteration 5/30... Test Accuracy - 0.7386 (65/88 correct predictions)
Iteration 6/30... Test Accuracy - 0.7386 (65/88 correct predictions)
Iteration 7/30... Test Accuracy - 0.7386 (65/88 correct predictions)
Iteration 8/30... Test Accuracy - 0.7386 (65/88 correct predictions)
Iteration 9/30... Test Accuracy - 0.7386 (65/88 correct predictions)
Iteration 10/30... Test Accuracy - 0.7386 (65/88 correct predictions)
Iteration 11/30... Test Accuracy - 0.7386 (65/88 correct predictions)
Iteration 12/30... Test Accuracy - 0.7386 (65/88 correct predictions)
Iteration 13/30... Test Accuracy - 0.7386 (65/88 correct predictions)
Iteration 14/30... Test Accuracy - 0.7386 (65/88 correct predictions)
Iteration 15/30... Test Accuracy - 0.7386 (65/88 correct predictions)
Iteration 16/30... Test Accuracy - 0.7386 (65/88 correct predictions)
Iteration 17/30... Test Accuracy - 0.7386 (65/88 correct predictions)
Iteration 18/30... Test Accuracy - 0.7386 (65/88 correct predictions)
Iteration 19/30... Test Accuracy - 0.7386 (65/88 correct predictions)
Iteration 20/30... Test Accuracy - 0.7386 (65/88 correct predictions)
Iteration 21/30... Test Accuracy - 0.7386 (65/88 correct predictions)
Iteration 22/30... Test Accuracy - 0.7386 (65/88 correct predictions)
Iteration 23/30... Test Accuracy - 0.7386 (65/88 correct predictions)
...
Mean Accuracy:          0.7386
Standard Deviation:     0.0000
95% Confidence Interval: [0.7386, 0.7386]
========================================================
```



Challenges

Feature Selection

Parameter Selection

Data Mapping

Circuit Architecture

Cost Minimisation

Noise Handling

Set up and run on QPU

Contributions

- **Adrian Șofariu – data embedding and model concept**
- **Sebastian Șoptelea – data embedding and model concept**
- **Rodica Ioana Lung – optimization and cost minimisation**
- **Gheorghe Mahu – testing and validation**
- **Tudor-Dan Mihoc – testing and presentation**

Thank you for your attention!