```
from ucimlrepo import fetch_ucirepo
import pandas as pd
# fetch dataset
statlog_heart = fetch_ucirepo(id=145)

# data (as pandas dataframes)
X = statlog_heart.data.features
y = statlog_heart.data.targets

print(X.head())
```

```
    age  sex  chest-pain  rest-bp  serum-chol  fasting-blood-sugar  \
0  70.0  1.0         4.0    130.0       322.0                  0.0
1  67.0  0.0         3.0    115.0       564.0                  0.0
2  57.0  1.0         2.0    124.0       261.0                  0.0
3  64.0  1.0         4.0    128.0       263.0                  0.0
4  74.0  0.0         2.0    120.0       269.0                  0.0

   electrocardiographic  max-heart-rate  angina  oldpeak  slope  \
0                   2.0           109.0     0.0      2.4    2.0
1                   2.0           160.0     0.0      1.6    2.0
2                   0.0           141.0     0.0      0.3    1.0
3                   0.0           105.0     1.0      0.2    2.0
4                   2.0           121.0     1.0      0.2    1.0

   major-vessels  thal
0            3.0   3.0
1            0.0   7.0
2            0.0   7.0
3            1.0   7.0
4            1.0   3.0
```

```
# metadata
print(statlog_heart.metadata)

# variable information
print(statlog_heart.variables)
```

```
X_encoded = pd.get_dummies(X, drop_first=True)
print(X_encoded.head())
```

```
    age  sex  chest-pain  rest-bp  serum-chol  fasting-blood-sugar  \
0  70.0  1.0         4.0    130.0       322.0                  0.0
1  67.0  0.0         3.0    115.0       564.0                  0.0
2  57.0  1.0         2.0    124.0       261.0                  0.0
3  64.0  1.0         4.0    128.0       263.0                  0.0
4  74.0  0.0         2.0    120.0       269.0                  0.0

   electrocardiographic  max-heart-rate  angina  oldpeak  slope  \
0                   2.0           109.0     0.0      2.4    2.0
```

```
1                   2.0           160.0     0.0      1.6    2.0
2                   0.0           141.0     0.0      0.3    1.0
3                   0.0           105.0     1.0      0.2    2.0
4                   2.0           121.0     1.0      0.2    1.0

   major-vessels  thal
0            3.0   3.0
1            0.0   7.0
2            0.0   7.0
3            1.0   7.0
4            1.0   3.0
```

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
```

```
encoder = OneHotEncoder(drop='first')  # drop='first' avoids multicollinearity
categorical_columns = ['chest-pain', 'electrocardiographic','thal']
column_transformer = ColumnTransformer(transformers=[('encoder', encoder, categorical_col
X_encoded = column_transformer.fit_transform(X)
X_encoded_df = pd.DataFrame(X_encoded)
print(X_encoded_df.head())
```

```
     0    1    2    3    4    5    6     7    8      9     10   11     12  \
0  0.0  0.0  1.0  0.0  1.0  0.0  0.0  70.0  1.0  130.0  322.0  0.0  109.0
1  0.0  1.0  0.0  0.0  1.0  0.0  1.0  67.0  0.0  115.0  564.0  0.0  160.0
2  1.0  0.0  0.0  0.0  0.0  0.0  1.0  57.0  1.0  124.0  261.0  0.0  141.0
3  0.0  0.0  1.0  0.0  0.0  0.0  1.0  64.0  1.0  128.0  263.0  0.0  105.0
4  1.0  0.0  0.0  0.0  1.0  0.0  0.0  74.0  0.0  120.0  269.0  0.0  121.0

    13   14   15   16
0  0.0  2.4  2.0  3.0
1  0.0  1.6  2.0  0.0
2  0.0  0.3  1.0  0.0
3  1.0  0.2  2.0  1.0
4  1.0  0.2  1.0  1.0
```

```
from sklearn.decomposition import PCA
```

```
print(f"Original feature shape: {X.shape}")
print(f"Target shape: {y.shape}")
```

```
# Apply PCA to reduce features from 13 to 8
pca = PCA(n_components=8)
X_reduced = pca.fit_transform(X_encoded)
```

```
X_pca_df = pd.DataFrame(X_reduced, columns=[f'PC{i+1}' for i in range(8)])
print(f"Reduced feature shape: {X_reduced.shape}")
```

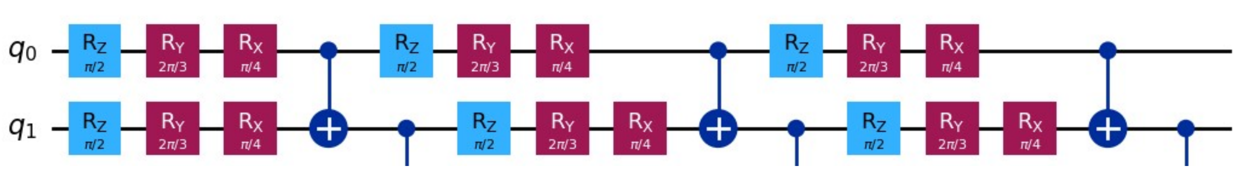
```
Original feature shape: (270, 13)
Target shape: (270, 1)
Reduced feature shape: (270, 8)
```

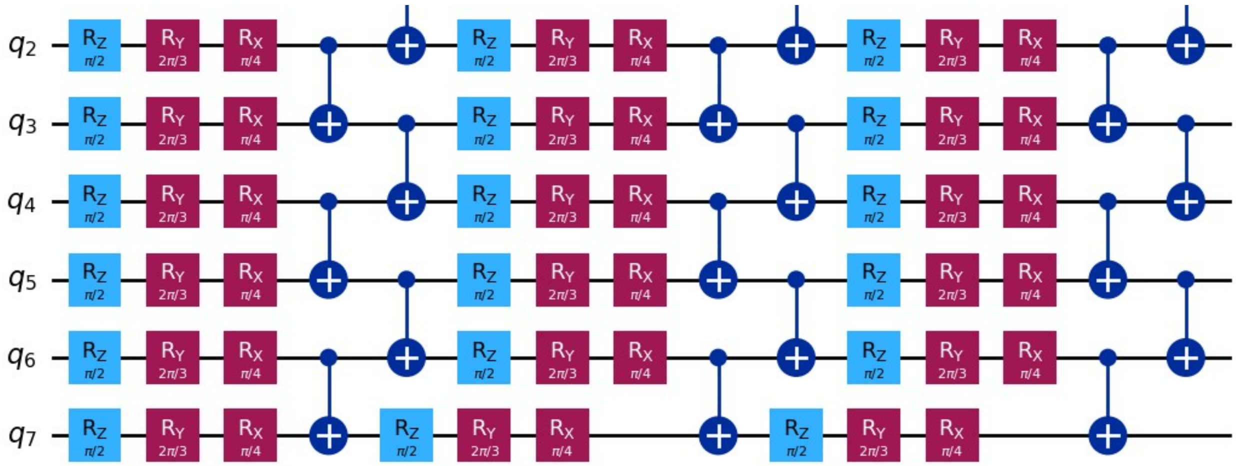
```
from qiskit.circuit.library import  ZZFeatureMap, PauliFeatureMap
def get_pauli(feature_dimension = 8, reps = 2):
    return PauliFeatureMap(feature_dimension=feature_dimension, paulis=['Z', 'YY'], reps=

def get_zzfeaturemap(feature_dimension = 8, reps = 2, entanglement = 'full'):
    return ZZFeatureMap(feature_dimension=feature_dimension, reps=reps, entanglement=enta
```

```
from sklearn.model_selection import train_test_split
# split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_reduced, y, test_size=0.20, random_
# split into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.125, rand
```

```
from qiskit import QuantumCircuit
import numpy as np

def custom_ansatz_two(num_qubits, layers, entanglement='pairwise', rotation_gates=None, e
    """
    Constructs a Two-Local ansatz circuit with fixed theta values.

    Parameters:
        num_qubits (int): The number of qubits in the circuit.
        layers (int): The number of rotation and entanglement layers.
        entanglement (str): The strategy for entangling qubits ('linear', 'circular', 'pa
        rotation_gates (list): List of rotation gates to use (e.g., ['ry', 'rz']).
        entanglement_gates (list): List of entanglement gates to use (e.g., ['cx']).

    Returns:
        QuantumCircuit: The constructed quantum circuit.
    """
    circuit = QuantumCircuit(num_qubits)

    # Define default rotation and entanglement gates if none provided
    if rotation_gates is None:
        rotation_gates = ['ry']
    if entanglement_gates is None:
        entanglement_gates = ['cx']  # Default entanglement gate
```

```
        entanglement_gates = ['cx']  # Default entanglement gate


    for layer in range(layers):
        # Apply rotation layer
        for qubit in range(num_qubits):
            for gate in rotation_gates:

                if gate == 'ry':
                    circuit.ry((np.pi/3)*2, qubit)
                elif gate == 'rz':
                    circuit.rz(np.pi/2, qubit)
                elif gate == 'rx':
                    circuit.rx(np.pi/4, qubit)

        if entanglement == 'linear':
            for i in range(num_qubits - 1):
                for gate in entanglement_gates:
                    if gate == 'cx':
                        circuit.cx(i, i + 1)
        elif entanglement == 'circular':
            for i in range(num_qubits):
                for gate in entanglement_gates:
                    if gate == 'cx':
                        circuit.cx(i, (i + 1) % num_qubits)  # Circular entanglement
        elif entanglement == 'pairwise':
            for i in range(0, num_qubits - 1, 2):  # Even indices
                for gate in entanglement_gates:
                    if gate == 'cx':
                        circuit.cx(i, i + 1)  # Entangle qubit i with i + 1
            for i in range(1, num_qubits - 1, 2):  # Odd indices
                for gate in entanglement_gates:
                    if gate == 'cx':
                        circuit.cx(i, i + 1)  # Entangle qubit i with i + 1


    return circuit

# Example:
num_qubits = 8
layers = 3
circuit = custom_ansatz_two(num_qubits, layers, rotation_gates=['rz','ry','rx'], entangle
circuit.draw(output='mpl')
```





```
from qiskit_machine_learning.algorithms.classifiers import VQC
from qiskit_algorithms.optimizers import COBYLA
from qiskit_algorithms.optimizers import SPSA

from qiskit_aer import QasmSimulator
from qiskit.primitives import BackendSampler

zz = get_zzfeaturemap()
backend = QasmSimulator()
```

```
import pandas as pd
from qiskit_machine_learning.algorithms.classifiers import VQC
from qiskit_algorithms.optimizers import COBYLA
from qiskit_aer import QasmSimulator
from qiskit.primitives import BackendSampler
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
import time

entanglement_strategies = ['linear', 'circular', 'pairwise']
backend = QasmSimulator()
```

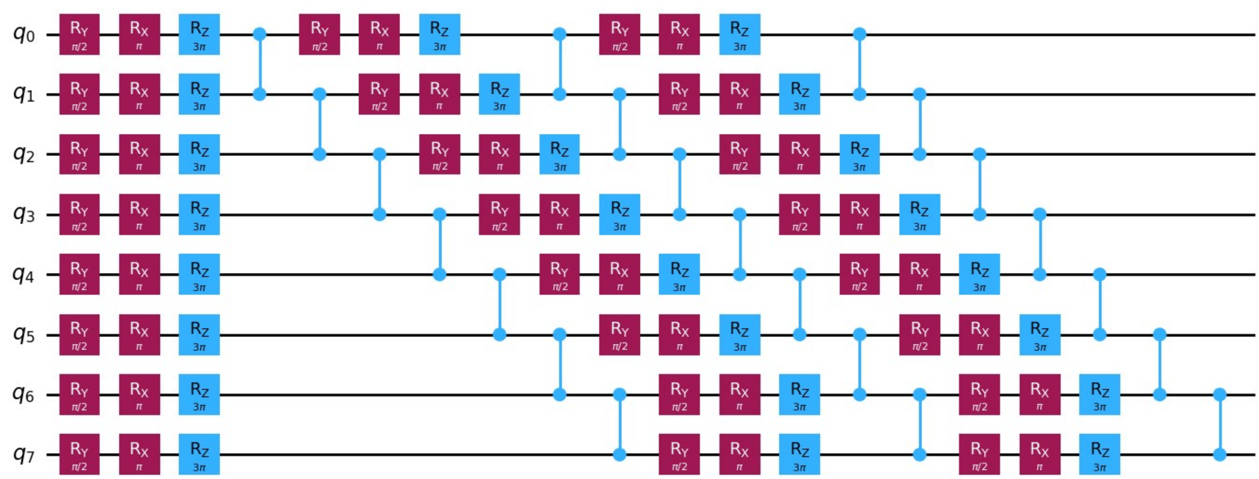
```
detailed_results = []
accuracy_results = []

# Loop over each entanglement strategy
for entanglement in entanglement_strategies:
    print(f"Testing entanglement strategy: {entanglement}")

    # Create ansatz with current entanglement strategy
    ansatz = custom_ansatz_two(num_qubits=8, layers=6, entanglement=entanglement, rotatio
    ansatz.draw(output='mpl')

    vqc = VQC(
        feature_map=get_zzfeaturemap(),
        ansatz=ansatz,
        optimizer=COBYLA(maxiter=500),
        sampler=BackendSampler(backend=backend)
    )

    start = time.time()
    vqc.fit(X_train, y_train.to_numpy())
    end = time.time()

    y_val_pred = vqc.predict(X_val)
    val_acc = accuracy_score(y_val, y_val_pred)
    val_f1 = f1_score(y_val, y_val_pred, average="weighted")
    val_precision = precision_score(y_val, y_val_pred, average="weighted")
    val_recall = recall_score(y_val, y_val_pred, average="weighted")

    y_test_pred = vqc.predict(X_test)
    test_acc = accuracy_score(y_test, y_test_pred)
    test_f1 = f1_score(y_test, y_test_pred, average="weighted")
    test_precision = precision_score(y_test, y_test_pred, average="weighted")
    test_recall = recall_score(y_test, y_test_pred, average="weighted")

    detailed_results.append({
        'Entanglement': entanglement,
        'Validation Accuracy': val_acc,
        'Validation F1 Score': val_f1,
        'Validation Precision': val_precision,
        'Validation Recall': val_recall,
        'Test Accuracy': test_acc,
        'Test F1 Score': test_f1,
        'Test Precision': test_precision,
        'Test Recall': test_recall,
        'Training Time (s)': end - start
    })

    accuracy_results.append({
        'Entanglement': entanglement,
        'Validation Accuracy': val_acc,
```

```
        'Test Accuracy': test_acc
    })

detailed_results_df = pd.DataFrame(detailed_results)

accuracy_results_df = pd.DataFrame(accuracy_results)

print("\nFinal Accuracy Results:")
print(accuracy_results_df)

print("\nDetailed Metrics Results:")
print(detailed_results_df, end=' ')


##GATES USED:
#2RY Gates, 2pi/3
```

```
Testing entanglement strategy: linear
Testing entanglement strategy: circular
Testing entanglement strategy: pairwise

Final Accuracy Results:
  Entanglement  Validation Accuracy  Test Accuracy
0       linear             0.481481       0.574074
1     circular             0.592593       0.481481
2     pairwise             0.629630       0.611111

Detailed Metrics Results:
  Entanglement  Validation Accuracy  Validation F1 Score  \
0       linear             0.481481             0.478632
1     circular             0.592593             0.590307
2     pairwise             0.629630             0.630647

   Validation Precision  Validation Recall  Test Accuracy  Test F1 Score  \
0              0.497475           0.481481       0.574074       0.574805
1              0.589646           0.592593       0.481481       0.482912
2              0.638584           0.629630       0.611111       0.607499

   Test Precision  Test Recall  Training Time (s)
0        0.585441     0.574074           2.179157
1        0.485653     0.481481           1.622357
2        0.639250     0.611111           1.565358
```

```
from qiskit import QuantumCircuit
import numpy as np


def custom_pauli(num_qubits):  #full entanglement
    circuit = QuantumCircuit(num_qubits)
    for layer in range(3):
        for qubit in range(num_qubits):
```

```
            circuit.ry(theta=np.pi/2, qubit=qubit)
            circuit.rx(theta=np.pi,qubit=qubit)
            circuit.rz(np.pi*3,qubit=qubit)

        # Entangle all qubits, full entanglement
        for i in range(num_qubits - 1):
            circuit.cz(i, i + 1)
    return circuit

ansatz_pauli = custom_pauli(num_qubits=8)
ansatz_pauli.draw(output='mpl')
```



```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
import time
```

```
pauli = get_pauli()
```

```
vqc_pauli = VQC(feature_map=pauli, ansatz=ansatz_pauli, optimizer=COBYLA(maxiter=500), samp


start = time.time()
vqc_pauli.fit(X_train, y_train.to_numpy())
end = time.time()

y_train_pred = vqc_pauli.predict(X_train)

y_val_pred = vqc_pauli.predict(X_val)

val_acc = accuracy_score(y_val, y_val_pred)
val_f1 = f1_score(y_val, y_val_pred, average="weighted")
val_precision = precision_score(y_val, y_val_pred, average="weighted")
val_recall = recall_score(y_val, y_val_pred, average="weighted")

print(f"Validation results: Accuracy = {val_acc:.2f}, F1 Score = {val_f1:.2f}, Precision =

y_test_pred = vqc_pauli.predict(X_test)

test_acc = accuracy_score(y_test, y_test_pred)
test_f1 = f1_score(y_test, y_test_pred, average="weighted")
test_precision = precision_score(y_test, y_test_pred, average="weighted")
test_recall = recall_score(y_test, y_test_pred, average="weighted")

print(f"Test results: Accuracy = {test_acc:.2f}, F1 Score = {test_f1:.2f}, Precision = {tes

print(f"Fitted vqc_pauli with training time = {end - start:.2f} seconds")
```

```
Validation results: Accuracy = 0.81, F1 Score = 0.81, Precision = 0.82, Recall = 0.81
Test results: Accuracy = 0.57, F1 Score = 0.58, Precision = 0.58, Recall = 0.57
Fitted vqc_pauli with training time = 1.76 seconds
```