```
from ucimlrepo import fetch_ucirepo
import pandas as pd
# fetch dataset
ionosphere = fetch_ucirepo(id=52)

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

# metadata
print(ionosphere.metadata)

# variable information
print(ionosphere.variables)


print(y['Class'].unique())
```

```
{'uci_id': 52, 'name': 'Ionosphere', 'repository_url': 'https://archive.ics.uci.edu/d
            name     role         type demographic description units  \
0    Attribute1  Feature   Continuous        None        None  None
1    Attribute2  Feature   Continuous        None        None  None
2    Attribute3  Feature   Continuous        None        None  None
3    Attribute4  Feature   Continuous        None        None  None
4    Attribute5  Feature   Continuous        None        None  None
5    Attribute6  Feature   Continuous        None        None  None
6    Attribute7  Feature   Continuous        None        None  None
7    Attribute8  Feature   Continuous        None        None  None
8    Attribute9  Feature   Continuous        None        None  None
9   Attribute10  Feature   Continuous        None        None  None
10  Attribute11  Feature   Continuous        None        None  None
11  Attribute12  Feature   Continuous        None        None  None
12  Attribute13  Feature   Continuous        None        None  None
13  Attribute14  Feature   Continuous        None        None  None
14  Attribute15  Feature   Continuous        None        None  None
15  Attribute16  Feature   Continuous        None        None  None
16  Attribute17  Feature   Continuous        None        None  None
17  Attribute18  Feature   Continuous        None        None  None
18  Attribute19  Feature   Continuous        None        None  None
19  Attribute20  Feature   Continuous        None        None  None
20  Attribute21  Feature   Continuous        None        None  None
21  Attribute22  Feature   Continuous        None        None  None
22  Attribute23  Feature   Continuous        None        None  None
23  Attribute24  Feature   Continuous        None        None  None
24  Attribute25  Feature   Continuous        None        None  None
25  Attribute26  Feature   Continuous        None        None  None
26  Attribute27  Feature   Continuous        None        None  None
27  Attribute28  Feature   Continuous        None        None  None
28  Attribute29  Feature   Continuous        None        None  None
29  Attribute30  Feature   Continuous        None        None  None
30  Attribute31  Feature   Continuous        None        None  None
31  Attribute32  Feature   Continuous        None        None  None
```

```
32  Attribute33  Feature   Continuous        None        None  None
33  Attribute34  Feature   Continuous        None        None  None
34        Class   Target  Categorical        None        None  None

   missing_values
0              no
1              no
2              no
3              no
4              no
5              no
6              no
7              no
8              no
9              no
10             no
11             no
12             no
13             no
14             no
15             no
16             no
17             no
18             no
```

```
from sklearn.decomposition import PCA #fitting 34 features into 12 qubits
```

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

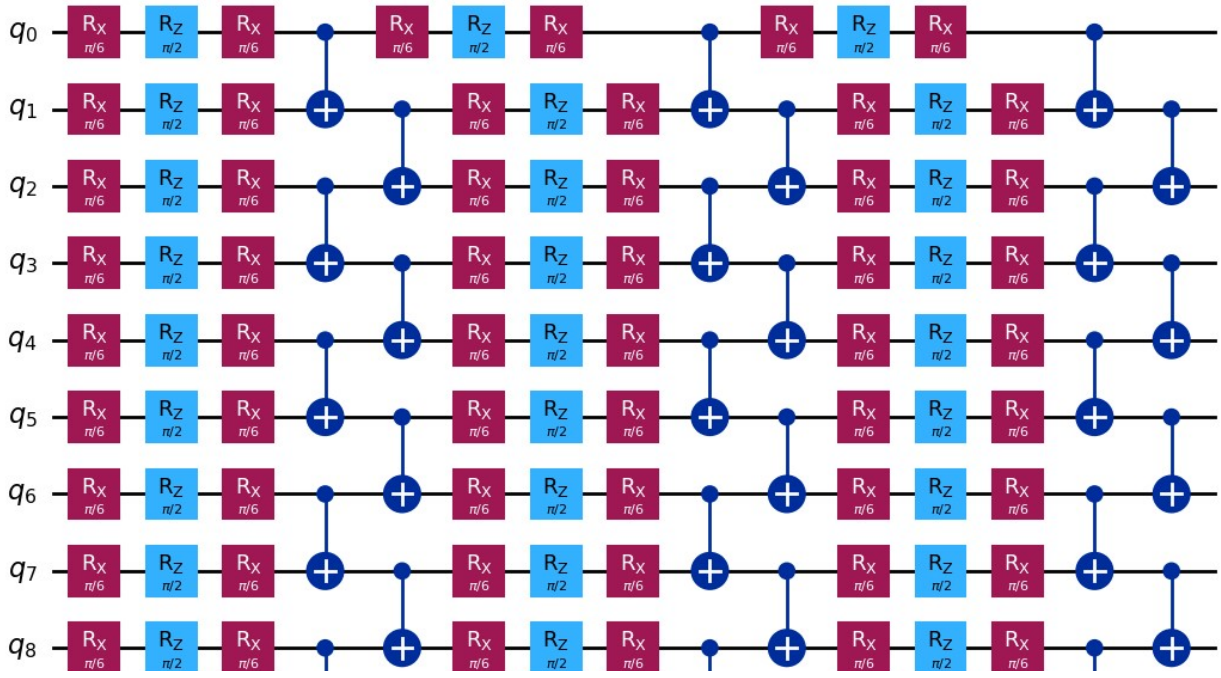
```
# Apply PCA to reduce features from 34 to 12
pca = PCA(n_components=12)
X_reduced = pca.fit_transform(X)

# Create a DataFrame for PCA results
X_pca_df = pd.DataFrame(X_reduced, columns=[f'PC{i+1}' for i in range(12)])
print(f"Reduced feature shape: {X_reduced.shape}")
```

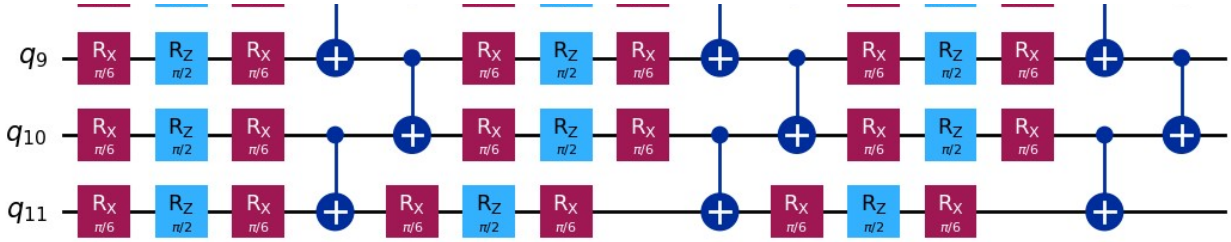
```
Original feature shape: (351, 34)
Target shape: (351, 1)
Reduced feature shape: (351, 12)
```

```
from qiskit.circuit.library import  ZZFeatureMap, PauliFeatureMap
def get_pauli(feature_dimension = 12, reps = 2):
```

```
    return PauliFeatureMap(feature_dimension=feature_dimension, paulis=['Z', 'YY'], reps=

def get_zzfeaturemap(feature_dimension = 12, 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)

    if rotation_gates is None:
        rotation_gates = ['ry']
    if entanglement_gates is None:
        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/4, qubit)
                elif gate == 'rz':
                    circuit.rz(np.pi/2, qubit)
                elif gate == 'rx':
                    circuit.rx(np.pi/6, qubit)
```

```
        # Apply entanglement layer using the specified entanglement strategy
        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 = 12
layers = 3
circuit = custom_ansatz_two(num_qubits, layers, rotation_gates=['rx','rz','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']

# Setup the quantum backend
```

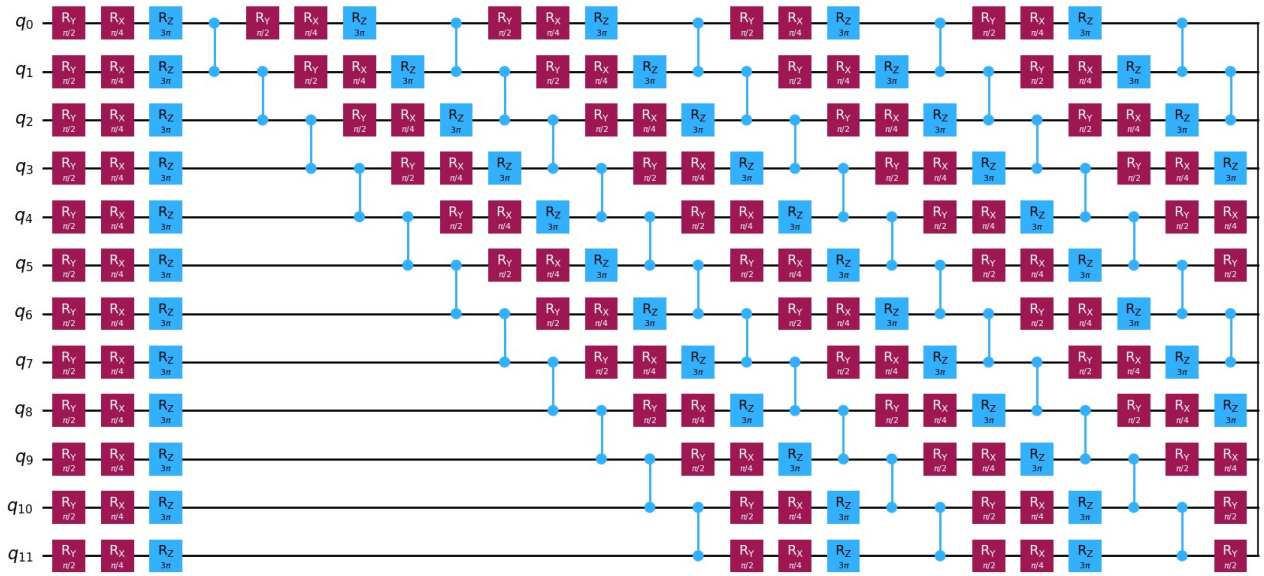
```
backend = QasmSimulator()

detailed_results = []
accuracy_results = []

for entanglement in entanglement_strategies:
    print(f"Testing entanglement strategy: {entanglement}")

    ansatz = custom_ansatz_two(num_qubits=12, layers=5, entanglement=entanglement, rotati
    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,
```

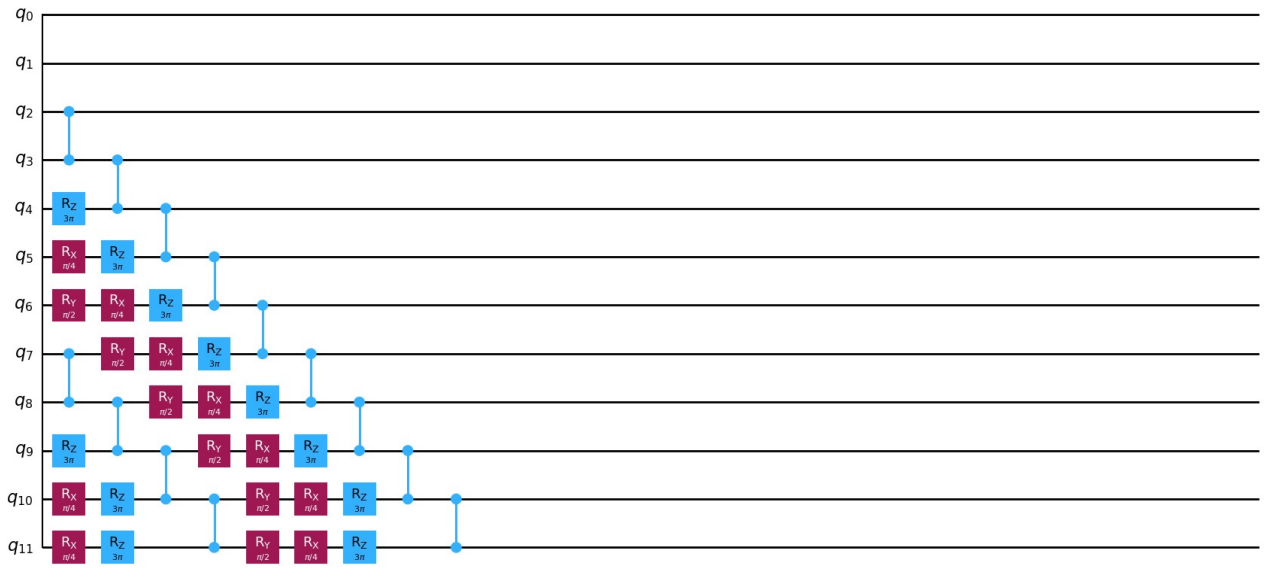
```
        'Test Accuracy': test_acc
    })

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:
#RY,RY,RY
#theta: ry: pi/4,
```

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

Final Accuracy Results:
  Entanglement  Validation Accuracy  Test Accuracy
0       linear             0.657143       0.492958
1     circular             0.600000       0.394366
2     pairwise             0.457143       0.436620

Detailed Metrics Results:
  Entanglement  Validation Accuracy  Validation F1 Score  \
0       linear             0.657143             0.657143
1     circular             0.600000             0.606667
2     pairwise             0.457143             0.460883

   Validation Precision  Validation Recall  Test Accuracy  Test F1 Score  \
0              0.657143           0.657143       0.492958       0.500213
1              0.628758           0.600000       0.394366       0.405257
2              0.465306           0.457143       0.436620       0.447148

   Test Precision  Test Recall  Training Time (s)
0        0.510349     0.492958           5.569705
1        0.452124     0.394366          17.819164
2        0.463403     0.436620           5.243273
```

```
from qiskit import QuantumCircuit
import numpy as np


def custom_pauli(num_qubits):  #full entanglement
    circuit = QuantumCircuit(num_qubits)
    for layer in range(5):
        for qubit in range(num_qubits):
```

```
            circuit.ry(theta=np.pi/2, qubit=qubit)
            circuit.rx(theta=np.pi/4,qubit=qubit)
            circuit.rz(np.pi*3,qubit=qubit)
        # Entangle all qubits, full entanglement, control Z gate
        for i in range(num_qubits - 1):
            circuit.cz(i, i + 1)
    return circuit

ansatz_pauli = custom_pauli(num_qubits=12)
#circuit:
ansatz_pauli.draw(output='mpl')
```





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


pauli = get_pauli()

# Initialize VQC with the Pauli ansatz
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 = 0.57, F1 Score = 0.57, Precision = 0.56, Recall = 0.57
Test results: Accuracy = 0.54, F1 Score = 0.52, Precision = 0.51, Recall = 0.54
Fitted vqc_pauli with training time = 7.42 seconds
```