Date:

TASK 9: Implement a QSVM on the Iris dataset using PennyLane

Aim: To implement a Quantum Support Vector Machine (QSVM) using PennyLane and scikit-learn, where the quantum kernel is constructed from a quantum feature map, and evaluate its performance on the Iris dataset for classification tasks.

1 Mathematical Model of the QSVM Algorithm

1. **Classical SVM Decision Function**

   The decision function for an SVM classifier is:

$$f(x) = sign\left(\sum_{i=1}^{N} \alpha_i\, y_i K(x_i, x) + \ b\right)$$

   where

   - x_i = training data,
   - y_i = class labels,
   - α_i = Lagrange multipliers,
   - $K(x_i, x)$ = kernel function,
   - b = bias term.

2. **Quantum Kernel (Fidelity Kernel)**

   In QSVM, the kernel is computed as the fidelity between two quantum states encoded by the feature map:

$$K(x, x') = |\langle \Phi(x) | \Phi(x') \rangle|^2$$

   where $|\Phi(x)\rangle$ is the quantum state obtained after applying the feature map circuit.

3. **Feature Map (Encoding)**

We embed classical features into quantum states using rotations and entangling gates. For each feature vector $x = \ (x_1, x_2, x_3, x_4)$.

$$|\Phi(x)\rangle = U_{\emptyset}(x)\big|0\rangle^{\otimes n}$$

where $U_{\emptyset}(x)$ consists of

- Hadamard gates (superposition)
- $RZ(x_i)$.rotations for feature encoding
- $CNOT + RZ$ entanglement (similar to ZZFeatureMap).

2 Algorithm - QSVM Algorithm

1. **Load dataset** (Iris, 150 samples, 3 classes).
2. **Preprocess**
   - Select features [sepal_length, sepal_width, petal_length, petal_width].
   - Encode target labels numerically.
   - Split dataset into train (67%) and test (33%).
3. **Quantum Feature Map**
   - Apply Hadamard (H) gates to all qubits.
   - Encode features into rotations $RZ(x_i)$.
   - Add entanglement with $CNOT + RZ\left(x_i . x_j\right)$.
4. **Quantum Kernel Construction**
   - Use kernel_circuit: apply $U_\emptyset(x)$, then adjoint$U_\emptyset(x')^\dagger$.
   - Measure overlap (fidelity).
5. **Train QSVM**
   - Compute kernel matrix for training data.
   - Train **SVC(kernel = "precomputed")** using scikit-learn.
6. **Test QSVM**
   - Compute test kernel matrix.
   - Predict labels for test set.
7. **Evaluate performance**
   - Confusion Matrix, Classification Report.
   - Prediction for new point (4.4, 4.4, 4.4, 4.4).

3 Program

```
#!pip install seaborn
#!pip install -U scikit-learn
#!pip install qiskit-algorithms
#!pip install qiskit-machine-learning
#!pip install pylatexenc
#!pip install pennylane

import pennylane as qml
from pennylane import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,
confusion_matrix
from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
```

```python
# ------------------------------
# Load Iris dataset
# ------------------------------
df_iris = pd.read_csv("iris.csv")
X = df_iris[['sepal.length', 'sepal.width', 'petal.length',
'petal.width']].values
y = df_iris['variety'].values

# Encode labels into integers
encoder = LabelEncoder()
y = encoder.fit_transform(y)

# Train-test split
x_train, x_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=42)

# ------------------------------
# Define Quantum Feature Map
# ------------------------------
n_qubits = 4
dev = qml.device("default.qubit", wires=n_qubits)

def feature_map(x):
    """Embedding classical features into quantum states"""
    for i in range(n_qubits):
        qml.Hadamard(wires=i)
        qml.RZ(x[i], wires=i)
    # Add entanglement (similar to ZZFeatureMap)
    for i in range(n_qubits - 1):
        qml.CNOT(wires=[i, i+1])
        qml.RZ((x[i] * x[i+1]), wires=i+1)
        qml.CNOT(wires=[i, i+1])

# Kernel evaluation circuit
@qml.qnode(dev)
def kernel_circuit(x1, x2):
    feature_map(x1)
    qml.adjoint(feature_map)(x2)
    return qml.probs(wires=range(n_qubits))

# ------------------------------
# Display Quantum Circuits
# ------------------------------
sample_x = x_train[0]
sample_y = x_train[1]
```

```python
# Draw feature map circuit
@qml.qnode(dev)
def feature_map_circuit(x):
    feature_map(x)
    return qml.state()

print("\n--- Feature Map Circuit ---")
print(qml.draw(feature_map_circuit)(sample_x))

# Draw kernel circuit
print("\n--- Kernel Circuit ---")
print(qml.draw(kernel_circuit)(sample_x, sample_y))

# Optional: matplotlib visualization
# Draw feature map circuit
print("\n--- Feature Map Circuit ---")
fig, ax = qml.draw_mpl(feature_map_circuit)(sample_x)
plt.show()

# Draw kernel circuit
print("\n--- Kernel Circuit ---")
fig, ax = qml.draw_mpl(kernel_circuit)(sample_x, sample_y)
plt.show()

# -------------------------------
# Construct Gram (Kernel) Matrices
# -------------------------------
def kernel(x1, x2):
    """Return fidelity between |Φ(x1)> and |Φ(x2)>"""
    return kernel_circuit(x1, x2)[0]

def compute_kernel_matrix(X1, X2):
    K = np.zeros((len(X1), len(X2)))
    for i, x1 in enumerate(X1):
        for j, x2 in enumerate(X2):
            K[i, j] = kernel(x1, x2)
    return K

K_train = compute_kernel_matrix(x_train, x_train)
K_test = compute_kernel_matrix(x_test, x_train)

# -------------------------------
# Train QSVM
# -------------------------------
qsvm_model = SVC(kernel="precomputed")
qsvm_model.fit(K_train, y_train)
```

```
# Predictions
y_pred = qsvm_model.predict(K_test)

print("\nConfusion Matrix")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report")
print(classification_report(y_test, y_pred,
target_names=encoder.classes_))

# -------------------------------
# Test on a new input
# -------------------------------
new_point = np.array([[4.4, 4.4, 4.4, 4.4]])
K_new = compute_kernel_matrix(new_point, x_train)
pred_label = qsvm_model.predict(K_new)
print("Predicted flower type for (4.4, 4.4, 4.4, 4.4):",
encoder.inverse_transform(pred_label)[0])
```

4 Result

The QSVM implemented with PennyLane successfully classifies the Iris dataset with high accuracy (~93%). The quantum kernel (fidelity-based) effectively maps classical features into higher-dimensional Hilbert space, enabling better separation of non-linear data.