# 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:

N

𝑓(𝑥) = 𝑠𝑖𝑔𝑛 (Σ αi 𝑦i𝐾(𝑥i, 𝑥) + 𝑏)

i=1

where

* + 𝑥i = training data,
  + 𝑦i = class labels,
  + αi = Lagrange multipliers,
  + 𝐾(𝑥i, 𝑥) = kernel function,
  + 𝑏 = bias term.

# 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') = |(Φ(𝑥)|Φ(𝑥')⟩|2

where |Φ(𝑥)⟩ is the quantum state obtained after applying the feature map circuit.

# Feature Map (Encoding)

We embed classical features into quantum states using rotations and entangling gates. For each feature vector 𝑥 = (𝑥1, 𝑥2, 𝑥3, 𝑥4).

|Φ(𝑥)⟩ = 𝑈∅(𝑥)|0⟩⊗n

where 𝑈∅(𝑥) consists of

* + Hadamard gates (superposition)
  + 𝑅𝑍(𝑥i).rotations for feature encoding
  + 𝐶𝑁𝑂𝑇 + 𝑅𝑍 entanglement (similar to ZZFeatureMap).

# 2 Algorithm - QSVM Algorithm

1. **Load dataset** (Iris, 150 samples, 3 classes).

# Preprocess

* + Select features [sepal\_length, sepal\_width, petal\_length, petal\_width].
  + Encode target labels numerically.
  + Split dataset into train (67%) and test (33%).

# Quantum Feature Map

* + Apply Hadamard (H) gates to all qubits.
  + Encode features into rotations 𝑅𝑍(𝑥i).
  + Add entanglement with 𝐶𝑁𝑂𝑇 + 𝑅𝑍(𝑥i. 𝑥j).

# Quantum Kernel Construction

* + Use kernel\_circuit: apply 𝑈∅(𝑥), then adjoint𝑈∅(𝑥')†.
  + Measure overlap (fidelity).

# Train QSVM

* + Compute kernel matrix for training data.
  + Train **SVC(kernel = “precomputed”)** using scikit-learn.

# Test QSVM

* + Compute test kernel matrix.
  + Predict labels for test set.

# 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

# -------------------------------

# Load Iris dataset

# -------------------------------

df\_iris = pd.read\_csv("/content/Iris.csv")

X = df\_iris[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm',

'PetalWidthCm']].values

y = df\_iris['Species'].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]

# 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])

**Outputs:**





