Assignment 3.: Implementation of Quantum Computing: Deuch algorithm and Quantum SVM

Dr. Suyong Eum

18^{th} July, 2024

1 Description

The aim of this assignment is to gain an in-depth understanding of the Deutsch algorithm through practical implementation, and to evaluate the limitations of the current Quantum Support Vector Machine (QSVM) implementation in comparison to a fully optimized classical Support Vector Machine (CSVM). The Deutsch algorithm showcases the power of quantum parallelism by solving specific problems more efficiently than classical algorithms. This task will include both theoretical study and hands-on implementation using quantum programming frameworks such as Qiskit. QSVM extends CSVM by utilizing quantum principles such as superposition and entanglement, which can potentially offer faster processing and improved handling of complex datasets. However, the practical performance of QSVMs often falls short of their theoretical promise. A key objective of this assignment is to experience firsthand the limitations of the current QSVM implementations when compared to fully optimized CSVMs.

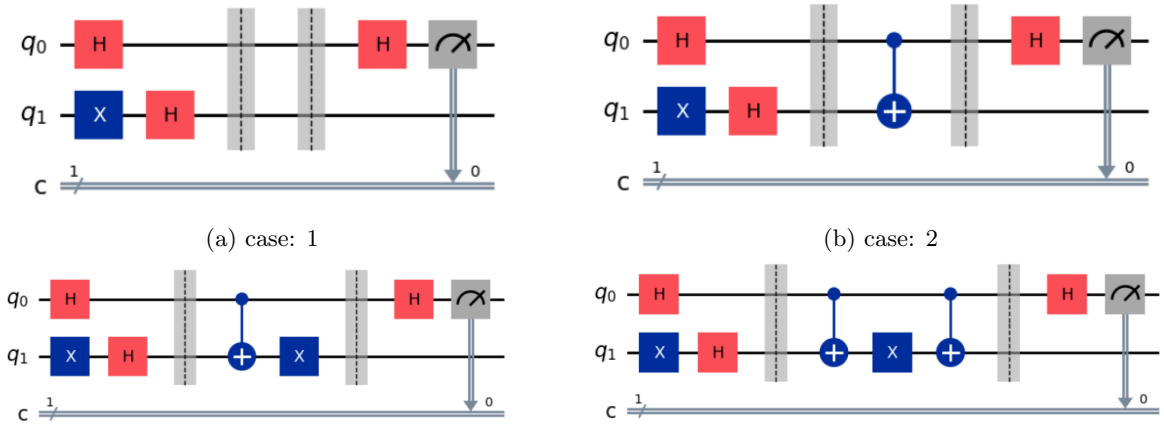
2 Required Tasks

This is a group assignment consisting of two main parts: one for implementing the Deutsch algorithm and the other for comparing the Quantum Support Vector Machine (QSVM) with the classical Support Vector Machine (CSVM).

2.1 Deutsch Algorithm Implementation

In this part, you are expected to create four quantum circuits as depicted in Fig.1. You should then discuss the differences among these circuits. The discussion should address the following questions:

1. What are the states of both qbits, q_0 and q_1, just before and after the last Hadamard gate at q_0 for all cases? Visualize the states of both cases using Bloch Sphere.
2. What is the measured value at the end of each circuit?



(a) case: 1 (b) case: 2

(c) case: 3 (d) case: 4

Figure 1: Implementation of Deutsch algorithm

The code, the implementation of one of the cases in Fig.1, is given in Appendix A to help you complete this assignment. You should be able to understand the code reading the comments provided. Please, extend the code to create all the quantum circuits above, and complete this assignment[1].

[1] You need to install qiskit. Please, refer the link; https://qiskit.org/documentation/getting_started.html.

2.2 QSVM and CSVM comparison

In the comparison between Quantum Support Vector Machine (QSVM) and classical Support Vector Machine (CSVM), you will evaluate their performance and accuracy. Since QSVM implementation is readily available in the Qiskit library, you will focus on utilizing existing functions rather than implementing QSVM from scratch. Here are the expected requirements for this part of your assignment:

1. Use the data set "from sklearn.datasets import load_digits".
2. Select two sets of data; for example, consider sets corresponding to numbers 3 and 4.
3. Reduce the dimensionality from 64 to two, and perform binary classification using QSVM and CSVM.
   (a) Compare the execution times between QSVM and CSVM.
   (b) Compare the accuracy of both approaches.
   (c) Visualize the decision boundaries and data points for both QSVM and CSVM.
4. Repeat the entire process as many times as needed,
   (a) with two different sets of data; for example, datasets corresponding to numbers 1 and 2.
   (b) with different kernel functions in CSVM.

Here is a reference for implementing QSVM:

1. https://qiskit-community.github.io/qiskit-machine-learning/tutorials/07_pegasos_qsvc.html

3 Administrative

- Oral update on progress will take place on July 25, 2024, during the class.
- Final report due: 24:00, July 31, 2024
  - Submission to (suyong@ist.osaka-u.ac.jp)
  - Please submit both the codes and the report.
  - When you send me an email, please make sure to include the Group number and assignment number in the title, for example, "Group 3: Assignment 3".

Appendices

A Implementation of Deutsch algorithm

```
##################################################
# Quantum circuit creation for Deutsch algorithm
##################################################
# qiskit     = 1.1.0
# qiskit-aer = 0.14.2
##################################################
from qiskit import *
from qiskit_aer import Aer
from qiskit.visualization import plot_histogram

# Number of quantum circuits
num_qc   = 2

# Number of classical circuits
num_cc   = 1

# Create a circuit object
qcircuit = QuantumCircuit(num_qc, num_cc)

# Add x gate to the quantum circuit (q_1)
qcircuit.x(1)

# Add Hadamard gate to the quantum circuits, (q_0) and (q_1), respectively
qcircuit.h(0)
qcircuit.h(1)

# Add CNOT gate which connects the quantum circuit from (q_0) to (q_1)
qcircuit.cx(0,1)

# Add Hadamard gate to the quantum circuit (q_0)
qcircuit.h(0)

# Measure the quantum circuit (q_0) and show the result in (C)
qcircuit.measure(0,0)

# Draw the quantum circuit which includes all the gates added
qcircuit.draw(output='mpl')

##################################################
# Run the circuit on a quantum simulator
##################################################
aer_sim = Aer.get_backend('aer_simulator')
t_qpe = transpile(qcircuit, aer_sim)
results = aer_sim.run(t_qpe, shots=1024).result()
answer = results.get_counts()
plot_histogram(answer)
```