QKM_verification_two_qubits

January 3, 2025

[1]:
```
!pip install qiskit-ibm-runtime
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: qiskit-ibm-runtime in
/home/josemiguel/.local/lib/python3.10/site-packages (0.27.0)
Requirement already satisfied: websocket-client>=1.5.1 in
/home/josemiguel/.local/lib/python3.10/site-packages (from qiskit-ibm-runtime)
(1.8.0)
Requirement already satisfied: python-dateutil>=2.8.0 in
/home/josemiguel/.local/lib/python3.10/site-packages (from qiskit-ibm-runtime)
(2.9.0.post0)
Requirement already satisfied: requests>=2.19 in
/home/josemiguel/.local/lib/python3.10/site-packages (from qiskit-ibm-runtime)
(2.32.3)
Requirement already satisfied: numpy>=1.13 in
/home/josemiguel/.local/lib/python3.10/site-packages (from qiskit-ibm-runtime)
(1.26.4)
Requirement already satisfied: urllib3>=1.21.1 in
/home/josemiguel/.local/lib/python3.10/site-packages (from qiskit-ibm-runtime)
(2.2.2)
Requirement already satisfied: requests-ntlm>=1.1.0 in
/home/josemiguel/.local/lib/python3.10/site-packages (from qiskit-ibm-runtime)
(1.3.0)
Requirement already satisfied: ibm-platform-services>=0.22.6 in
/home/josemiguel/.local/lib/python3.10/site-packages (from qiskit-ibm-runtime)
(0.55.3)
Requirement already satisfied: pydantic>=2.5.0 in
/home/josemiguel/.local/lib/python3.10/site-packages (from qiskit-ibm-runtime)
(2.8.2)
Requirement already satisfied: qiskit>=1.1.0 in
/home/josemiguel/.local/lib/python3.10/site-packages (from qiskit-ibm-runtime)
(1.1.2)
Requirement already satisfied: ibm-cloud-sdk-core<4.0.0,>=3.20.6 in
/home/josemiguel/.local/lib/python3.10/site-packages (from ibm-platform-
services>=0.22.6->qiskit-ibm-runtime) (3.20.6)
Requirement already satisfied: annotated-types>=0.4.0 in
/home/josemiguel/.local/lib/python3.10/site-packages (from
pydantic>=2.5.0->qiskit-ibm-runtime) (0.7.0)
```

```
Requirement already satisfied: pydantic-core==2.20.1 in
/home/josemiguel/.local/lib/python3.10/site-packages (from
pydantic>=2.5.0->qiskit-ibm-runtime) (2.20.1)
Requirement already satisfied: typing-extensions>=4.6.1 in
/home/josemiguel/.local/lib/python3.10/site-packages (from
pydantic>=2.5.0->qiskit-ibm-runtime) (4.12.2)
Requirement already satisfied: six>=1.5 in /usr/lib/python3/dist-packages (from
python-dateutil>=2.8.0->qiskit-ibm-runtime) (1.16.0)
Requirement already satisfied: symengine>=0.11 in
/home/josemiguel/.local/lib/python3.10/site-packages (from
qiskit>=1.1.0->qiskit-ibm-runtime) (0.11.0)
Requirement already satisfied: scipy>=1.5 in
/home/josemiguel/.local/lib/python3.10/site-packages (from
qiskit>=1.1.0->qiskit-ibm-runtime) (1.14.0)
Requirement already satisfied: sympy>=1.3 in
/home/josemiguel/.local/lib/python3.10/site-packages (from
qiskit>=1.1.0->qiskit-ibm-runtime) (1.13.1)
Requirement already satisfied: stevedore>=3.0.0 in
/home/josemiguel/.local/lib/python3.10/site-packages (from
qiskit>=1.1.0->qiskit-ibm-runtime) (5.2.0)
Requirement already satisfied: rustworkx>=0.14.0 in
/home/josemiguel/.local/lib/python3.10/site-packages (from
qiskit>=1.1.0->qiskit-ibm-runtime) (0.15.1)
Requirement already satisfied: dill>=0.3 in
/home/josemiguel/.local/lib/python3.10/site-packages (from
qiskit>=1.1.0->qiskit-ibm-runtime) (0.3.8)
Requirement already satisfied: idna<4,>=2.5 in /usr/lib/python3/dist-packages
(from requests>=2.19->qiskit-ibm-runtime) (3.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/lib/python3/dist-
packages (from requests>=2.19->qiskit-ibm-runtime) (2020.6.20)
Requirement already satisfied: charset-normalizer<4,>=2 in
/home/josemiguel/.local/lib/python3.10/site-packages (from
requests>=2.19->qiskit-ibm-runtime) (3.3.2)
Requirement already satisfied: pyspnego>=0.4.0 in
/home/josemiguel/.local/lib/python3.10/site-packages (from requests-
ntlm>=1.1.0->qiskit-ibm-runtime) (0.11.1)
Requirement already satisfied: cryptography>=1.3 in /usr/lib/python3/dist-
packages (from requests-ntlm>=1.1.0->qiskit-ibm-runtime) (3.4.8)
Requirement already satisfied: PyJWT<3.0.0,>=2.8.0 in
/home/josemiguel/.local/lib/python3.10/site-packages (from ibm-cloud-sdk-
core<4.0.0,>=3.20.6->ibm-platform-services>=0.22.6->qiskit-ibm-runtime) (2.9.0)
Requirement already satisfied: pbr!=2.1.0,>=2.0.0 in
/home/josemiguel/.local/lib/python3.10/site-packages (from
stevedore>=3.0.0->qiskit>=1.1.0->qiskit-ibm-runtime) (6.0.0)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/home/josemiguel/.local/lib/python3.10/site-packages (from
sympy>=1.3->qiskit>=1.1.0->qiskit-ibm-runtime) (1.3.0)
```

[2]:
```
!pip install qiskit
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: qiskit in
/home/josemiguel/.local/lib/python3.10/site-packages (1.1.2)
Requirement already satisfied: python-dateutil>=2.8.0 in
/home/josemiguel/.local/lib/python3.10/site-packages (from qiskit) (2.9.0.post0)
Requirement already satisfied: numpy<3,>=1.17 in
/home/josemiguel/.local/lib/python3.10/site-packages (from qiskit) (1.26.4)
Requirement already satisfied: dill>=0.3 in
/home/josemiguel/.local/lib/python3.10/site-packages (from qiskit) (0.3.8)
Requirement already satisfied: symengine>=0.11 in
/home/josemiguel/.local/lib/python3.10/site-packages (from qiskit) (0.11.0)
Requirement already satisfied: sympy>=1.3 in
/home/josemiguel/.local/lib/python3.10/site-packages (from qiskit) (1.13.1)
Requirement already satisfied: typing-extensions in
/home/josemiguel/.local/lib/python3.10/site-packages (from qiskit) (4.12.2)
Requirement already satisfied: rustworkx>=0.14.0 in
/home/josemiguel/.local/lib/python3.10/site-packages (from qiskit) (0.15.1)
Requirement already satisfied: stevedore>=3.0.0 in
/home/josemiguel/.local/lib/python3.10/site-packages (from qiskit) (5.2.0)
Requirement already satisfied: scipy>=1.5 in
/home/josemiguel/.local/lib/python3.10/site-packages (from qiskit) (1.14.0)
Requirement already satisfied: six>=1.5 in /usr/lib/python3/dist-packages (from
python-dateutil>=2.8.0->qiskit) (1.16.0)
Requirement already satisfied: pbr!=2.1.0,>=2.0.0 in
/home/josemiguel/.local/lib/python3.10/site-packages (from
stevedore>=3.0.0->qiskit) (6.0.0)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/home/josemiguel/.local/lib/python3.10/site-packages (from sympy>=1.3->qiskit)
(1.3.0)
```

[3]:
```
pip install qiskit_algorithms
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: qiskit_algorithms in
/home/josemiguel/.local/lib/python3.10/site-packages (0.3.0)
Requirement already satisfied: scipy>=1.4 in
/home/josemiguel/.local/lib/python3.10/site-packages (from qiskit_algorithms)
(1.14.0)
Requirement already satisfied: qiskit>=0.44 in
/home/josemiguel/.local/lib/python3.10/site-packages (from qiskit_algorithms)
(1.1.2)
Requirement already satisfied: numpy>=1.17 in
/home/josemiguel/.local/lib/python3.10/site-packages (from qiskit_algorithms)
(1.26.4)
Requirement already satisfied: symengine>=0.11 in
/home/josemiguel/.local/lib/python3.10/site-packages (from
```

```
qiskit>=0.44->qiskit_algorithms) (0.11.0)
Requirement already satisfied: stevedore>=3.0.0 in
/home/josemiguel/.local/lib/python3.10/site-packages (from
qiskit>=0.44->qiskit_algorithms) (5.2.0)
Requirement already satisfied: typing-extensions in
/home/josemiguel/.local/lib/python3.10/site-packages (from
qiskit>=0.44->qiskit_algorithms) (4.12.2)
Requirement already satisfied: dill>=0.3 in
/home/josemiguel/.local/lib/python3.10/site-packages (from
qiskit>=0.44->qiskit_algorithms) (0.3.8)
Requirement already satisfied: python-dateutil>=2.8.0 in
/home/josemiguel/.local/lib/python3.10/site-packages (from
qiskit>=0.44->qiskit_algorithms) (2.9.0.post0)
Requirement already satisfied: sympy>=1.3 in
/home/josemiguel/.local/lib/python3.10/site-packages (from
qiskit>=0.44->qiskit_algorithms) (1.13.1)
Requirement already satisfied: rustworkx>=0.14.0 in
/home/josemiguel/.local/lib/python3.10/site-packages (from
qiskit>=0.44->qiskit_algorithms) (0.15.1)
Requirement already satisfied: six>=1.5 in /usr/lib/python3/dist-packages (from
python-dateutil>=2.8.0->qiskit>=0.44->qiskit_algorithms) (1.16.0)
Requirement already satisfied: pbr!=2.1.0,>=2.0.0 in
/home/josemiguel/.local/lib/python3.10/site-packages (from
stevedore>=3.0.0->qiskit>=0.44->qiskit_algorithms) (6.0.0)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/home/josemiguel/.local/lib/python3.10/site-packages (from
sympy>=1.3->qiskit>=0.44->qiskit_algorithms) (1.3.0)
Note: you may need to restart the kernel to use updated packages.
```

[4]:
```
!pip install qiskit-aer
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: qiskit-aer in
/home/josemiguel/.local/lib/python3.10/site-packages (0.14.2)
Requirement already satisfied: qiskit>=0.45.2 in
/home/josemiguel/.local/lib/python3.10/site-packages (from qiskit-aer) (1.1.2)
Requirement already satisfied: psutil>=5 in
/home/josemiguel/.local/lib/python3.10/site-packages (from qiskit-aer) (6.0.0)
Requirement already satisfied: numpy>=1.16.3 in
/home/josemiguel/.local/lib/python3.10/site-packages (from qiskit-aer) (1.26.4)
Requirement already satisfied: scipy>=1.0 in
/home/josemiguel/.local/lib/python3.10/site-packages (from qiskit-aer) (1.14.0)
Requirement already satisfied: sympy>=1.3 in
/home/josemiguel/.local/lib/python3.10/site-packages (from
qiskit>=0.45.2->qiskit-aer) (1.13.1)
Requirement already satisfied: dill>=0.3 in
/home/josemiguel/.local/lib/python3.10/site-packages (from
qiskit>=0.45.2->qiskit-aer) (0.3.8)
```

```
Requirement already satisfied: python-dateutil>=2.8.0 in
/home/josemiguel/.local/lib/python3.10/site-packages (from
qiskit>=0.45.2->qiskit-aer) (2.9.0.post0)
Requirement already satisfied: typing-extensions in
/home/josemiguel/.local/lib/python3.10/site-packages (from
qiskit>=0.45.2->qiskit-aer) (4.12.2)
Requirement already satisfied: rustworkx>=0.14.0 in
/home/josemiguel/.local/lib/python3.10/site-packages (from
qiskit>=0.45.2->qiskit-aer) (0.15.1)
Requirement already satisfied: symengine>=0.11 in
/home/josemiguel/.local/lib/python3.10/site-packages (from
qiskit>=0.45.2->qiskit-aer) (0.11.0)
Requirement already satisfied: stevedore>=3.0.0 in
/home/josemiguel/.local/lib/python3.10/site-packages (from
qiskit>=0.45.2->qiskit-aer) (5.2.0)
Requirement already satisfied: six>=1.5 in /usr/lib/python3/dist-packages (from
python-dateutil>=2.8.0->qiskit>=0.45.2->qiskit-aer) (1.16.0)
Requirement already satisfied: pbr!=2.1.0,>=2.0.0 in
/home/josemiguel/.local/lib/python3.10/site-packages (from
stevedore>=3.0.0->qiskit>=0.45.2->qiskit-aer) (6.0.0)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/home/josemiguel/.local/lib/python3.10/site-packages (from
sympy>=1.3->qiskit>=0.45.2->qiskit-aer) (1.3.0)
```

[5]:
```
!pip install pylatexenc
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: pylatexenc in
/home/josemiguel/.local/lib/python3.10/site-packages (2.10)
```

[ ]:
```
from qiskit_aer import AerSimulator,Aer# ojo son distintos AerSimulator y Aer
 ↪AerSimulator es para entrar en estadisticas de IBM
import pennylane as qml
from pennylane import numpy as np
import matplotlib.pyplot as plt
from time import time
from sklearn import svm
import scipy
```

[6]:
```
get_ipython().system('pip install pennylane')
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: pennylane in
/home/josemiguel/.local/lib/python3.10/site-packages (0.37.0)
Requirement already satisfied: networkx in
/home/josemiguel/.local/lib/python3.10/site-packages (from pennylane) (3.3)
Requirement already satisfied: toml in
/home/josemiguel/.local/lib/python3.10/site-packages (from pennylane) (0.10.2)
```

```
Requirement already satisfied: appdirs in
/home/josemiguel/.local/lib/python3.10/site-packages (from pennylane) (1.4.4)
Requirement already satisfied: scipy in
/home/josemiguel/.local/lib/python3.10/site-packages (from pennylane) (1.14.0)
Requirement already satisfied: rustworkx in
/home/josemiguel/.local/lib/python3.10/site-packages (from pennylane) (0.15.1)
Requirement already satisfied: pennylane-lightning>=0.37 in
/home/josemiguel/.local/lib/python3.10/site-packages (from pennylane) (0.37.0)
Requirement already satisfied: numpy<2.0 in
/home/josemiguel/.local/lib/python3.10/site-packages (from pennylane) (1.26.4)
Requirement already satisfied: typing-extensions in
/home/josemiguel/.local/lib/python3.10/site-packages (from pennylane) (4.12.2)
Requirement already satisfied: packaging in
/home/josemiguel/.local/lib/python3.10/site-packages (from pennylane) (24.1)
Requirement already satisfied: semantic-version>=2.7 in
/home/josemiguel/.local/lib/python3.10/site-packages (from pennylane) (2.10.0)
Requirement already satisfied: requests in
/home/josemiguel/.local/lib/python3.10/site-packages (from pennylane) (2.32.3)
Requirement already satisfied: autograd in
/home/josemiguel/.local/lib/python3.10/site-packages (from pennylane) (1.7.0)
Requirement already satisfied: cachetools in
/home/josemiguel/.local/lib/python3.10/site-packages (from pennylane) (5.5.0)
Requirement already satisfied: autoray>=0.6.11 in
/home/josemiguel/.local/lib/python3.10/site-packages (from pennylane) (0.6.12)
Requirement already satisfied: certifi>=2017.4.17 in /usr/lib/python3/dist-
packages (from requests->pennylane) (2020.6.20)
Requirement already satisfied: charset-normalizer<4,>=2 in
/home/josemiguel/.local/lib/python3.10/site-packages (from requests->pennylane)
(3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/lib/python3/dist-packages
(from requests->pennylane) (3.3)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/home/josemiguel/.local/lib/python3.10/site-packages (from requests->pennylane)
(2.2.2)
```

[2]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle

import warnings
warnings.filterwarnings("ignore")
# constants
n = 2
```

```
RANDOM_STATE = 42
LR = 1e-3
class_labels = ['0', '1']

def normalizeData(DATA_PATH = "./FEATURE_RESULTS/FEATURE_resultante_DP_ruptura.
 ↪csv"):
    """
    Normalizes the data
    """
    # Reads the data
    data = pd.read_csv(DATA_PATH)
    data = shuffle(data, random_state=RANDOM_STATE)
    X, Y = data[['area_pixels', ' mean_coords_x']].values, data[' class'].values

    # normalize the data
    scaler = MinMaxScaler(feature_range=(-0 * np.pi, 2 * np.pi))
    X = scaler.fit_transform(X)
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,␣
 ↪random_state=RANDOM_STATE)
    return X_train, X_test, Y_train, Y_test




import qiskit_algorithms
from qiskit_algorithms.optimizers import SPSA

from qiskit import QuantumCircuit

from qiskit.circuit.library import ZZFeatureMap, RealAmplitudes
from qiskit.quantum_info import Statevector

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

from sklearn.utils import shuffle

import warnings
warnings.filterwarnings("ignore")

TRAIN_DATA, TEST_DATA, TRAIN_LABELS, TEST_LABELS = normalizeData()
# Replace all occurrences of 2 with 1
```

```
TRAIN_LABELS = np.where(TRAIN_LABELS == 2, 1, TRAIN_LABELS)
TEST_LABELS = np.where(TEST_LABELS == 2, 1, TEST_LABELS)
#print(TRAIN_DATA)
```

[9]:
```
#

from qiskit_aer import AerSimulator,Aer # eye,  are different AerSimulator and␣
 ↪Aer AerSimulator is to enter IBM statistics
#import pennylane as qml
#from pennylane import numpy as np
import matplotlib.pyplot as plt
from time import time
from sklearn import svm
import scipy
```

[41]:
```
print(TRAIN_LABELS)
print(f"Dimensiones de TRAIN_LABELS: {TRAIN_LABELS.shape}")
```

```
[1 0 1 1 1 0 0 0 1 0 1 0 0 1 1 1 1 0 1 1 1 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0
 0 1 1 1 1 0 0 1 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 1 0 1 0 0 1 1 1 1 0 0 0 0
 1 0 1 0 0 0 0 1 0 0 1 0 0 0 0 0 1 1 1 1 1 0 1 1 0 0 0 1 1 0 0 0 0 1 0 0 1
 0 0 1 0 1 1 1 1 1 1 0 0 0 0 1 1 0 1 0 1 1 1 0 0 1 0 1 0 0 0 0 1 1 1 0 1 0
 0 1 1 0 1 0 1 1 1 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0 1 0 1 0
 1 0 0 0 1 0 1 1 1 0 0 0 1 0 1 1 1 1 1 1 0 0 1 0 1 0 1 1 1 0 0 1 1 0 0 0 1
 0 0 1 0 0 1 1 0 1 0 0 1 1 1 1 1 0 0 0 1 1 0 1 0 0 1 0 1 0 1 0 0 1 1 0 0 0
 1 0 1 1 0 0 1 1 0 1 0 0 1 1 0 0 1 0 1 0 1 1 0 0 1 1 0 1 0 1 1 1 0 1 1 0 1
 1 1 1 1 0 1 0 1 0 1 1 0 0 0 1 0 1 0 0 0 0 1 0 0 1 0 1 1 0 1 1 0 1 1 0 0 1
 0 0 1 0 0 0 0 0 1 1 0 1 1 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1
 0 1 0 0 0 0 1 1 0 0 1 0 0 1 1 0 0 0 1 0 0 0 0 1 1 1 0 0 0 0 1 0 1 0 0 1 0
 0 0 1 1 0 1 0 0 0 0 1 1 1 0 1 1 0 0 1 1 1 0 1 1 0 1 0 0 1 0 1 1 1 0 1 0 0
 1 0 1 1 1 0 1 1 0 1 0 0 1 1 1 0 1 0 1 1 0 0 0 1 0 0 1 1 0 1 0 0 0 1 0 0 0
 1 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 0 1 0 0 1 0 0 0 0 0 0 0 1 0
 0 0 1 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0]
Dimensiones de TRAIN_LABELS: (540,)
```

[11]:
```
# FIRST METHOD FOR CALCULATING THE KERNEL WITH STATE VECTOR
import numpy as np
from qiskit import QuantumCircuit, transpile, assemble
import matplotlib.pyplot as plt
from qiskit_aer import AerSimulator
from qiskit.quantum_info import Statevector
```

[12]:
```
def feature_map(circuit, x):
    # ZZMap modificado
    for i in range(2):
        circuit.h(i)
    circuit.rz(2 * x[0], 0)
```

```
    circuit.rz(2 * x[1], 1)
    circuit.cx(0, 1)
    circuit.rz(2 * (np.pi - x[0]) * (np.pi - x[1]), 1)
    circuit.cx(0, 1)

def inverse_feature_map(circuit, x):
    # Invertir ZZMap
    circuit.cx(0, 1)
    circuit.rz(-2 * (np.pi - x[0]) * (np.pi - x[1]), 1)
    circuit.cx(0, 1)
    circuit.rz(-2 * x[1], 1)
    circuit.rz(-2 * x[0], 0)
    for i in range(2):
        circuit.h(i)

def scalar_product1(circuit, x, y):
    feature_map(circuit, y)
    inverse_feature_map(circuit, x)

    backend = Aer.get_backend('statevector_simulator')
    t_qc = transpile(circuit, backend)
    qobj = assemble(t_qc)
    result = backend.run(qobj).result()
    statevector = result.get_statevector(circuit)
    probs = np.abs(statevector)**2
    return probs

def scalar_product(circuit, x, y):
    feature_map(circuit, y)
    inverse_feature_map(circuit, x)
    #backend = AerSimulator('statevector_simulator')
    backend = Aer.get_backend('statevector_simulator')
    t_qc = transpile(circuit, backend)
    qobj = assemble(t_qc)
    result = backend.run(qobj).result()
    statevector = result.get_statevector(circuit)
    probs = np.abs(statevector)**2
    return probs




def kernel_gram_matrix_full(X1, X2):
    print("Calculating Gram matrix")

    gram_matrix = np.zeros((X1.shape[0], X2.shape[0]))
    for i, x1 in enumerate(X1):
```

```
        print(int(i / len(X1) * 100), "%")
        for j, x2 in enumerate(X2):
            qc = QuantumCircuit(2, 2)
            x1 = x1.flatten()
            x2 = x2.flatten()
            prob = scalar_product(qc, x1, x2)[0]  #Use only the probability of␣
 ↪state '00'
            gram_matrix[i, j] = prob

    return gram_matrix
```

[13]:
```
def scalar_product(circuit, x, y):
    feature_map(circuit, y)
    inverse_feature_map(circuit, x)
    #backend = AerSimulator('statevector_simulator')
    backend = Aer.get_backend('statevector_simulator')
    t_qc = transpile(circuit, backend)
    qobj = assemble(t_qc)
    result = backend.run(qobj).result()
    statevector = result.get_statevector(circuit)
    probs = np.abs(statevector)**2
    return probs
```

[ ]:
```

```

[ ]:
```
#If the matrix is square, this is easier.
# the dimension of x1 and x2 are equal


def kernel_gram_matrix_full_symmetric(X1, X2):
    print("Calculando matriz de Gram")

    num_samples_X1 = X1.shape[0]


    gram_matrix = np.zeros((num_samples_X1, num_samples_X1))  # Make sure you␣
 ↪have the correct size

    # Calculate only the upper half of the matrix
    for i in range(num_samples_X1):
        print(int(i / num_samples_X1 * 100), "%")
        for j in range(i, num_samples_X1):  # Only calculate the top half
            qc = QuantumCircuit(2, 2)
            x1 = X1[i].flatten()
            x2 = X2[j].flatten()

            prob = scalar_product(qc, x1, x2)[0]
```

```
            gram_matrix[i, j] = prob
            gram_matrix[j, i] = prob  # Symmetry: assigning the same value to␣
 ↪the lower half

    return gram_matrix
x_train=TRAIN_DATA[:]
import time
inicio_tiempo = time.time()
matrix_simetrica = kernel_gram_matrix_full_symmetric(x_train, x_train)
fin=time.time()
tiempo=inicio_tiempo-fin
print(tiempo)
```

```
Calculando matriz de Gram
0 %
0 %
0 %
0 %
0 %
0 %
1 %
1 %
1 %
1 %
1 %
2 %
2 %
2 %
2 %
2 %
2 %
2 %
3 %
3 %
3 %
3 %
3 %
4 %
4 %
4 %
4 %
4 %
5 %
5 %
5 %
5 %
5 %
5 %
```

6 %
6 %
6 %
6 %
6 %
7 %
7 %
7 %
7 %
7 %
7 %
8 %
8 %
8 %
8 %
8 %
9 %
9 %
9 %
9 %
9 %
10 %
10 %
10 %
10 %
10 %
10 %
11 %
11 %
11 %
11 %
11 %
12 %
12 %
12 %
12 %
12 %
12 %
13 %
13 %
13 %
13 %
13 %
14 %
14 %
14 %
14 %
14 %

15 %
15 %
15 %
15 %
15 %
15 %
16 %
16 %
16 %
16 %
16 %
16 %
17 %
17 %
17 %
17 %
17 %
17 %
18 %
18 %
18 %
18 %
18 %
18 %
19 %
19 %
19 %
19 %
19 %
20 %
20 %
20 %
20 %
20 %
20 %
21 %
21 %
21 %
21 %
21 %
22 %
22 %
22 %
22 %
22 %
22 %
23 %
23 %
23 %
23 %

23 %
24 %
24 %
24 %
24 %
24 %
25 %
25 %
25 %
25 %
25 %
25 %
26 %
26 %
26 %
26 %
26 %
26 %
27 %
27 %
27 %
27 %
27 %
27 %
28 %
28 %
28 %
28 %
28 %
29 %
29 %
29 %
29 %
29 %
30 %
30 %
30 %
30 %
30 %
30 %
31 %
31 %
31 %
31 %
31 %
32 %
32 %
32 %
32 %

32 %
32 %
33 %
33 %
33 %
33 %
33 %
33 %
34 %
34 %
34 %
34 %
34 %
34 %
35 %
35 %
35 %
35 %
35 %
35 %
35 %
36 %
36 %
36 %
36 %
36 %
36 %
37 %
37 %
37 %
37 %
37 %
37 %
37 %
38 %
38 %
38 %
38 %
38 %
38 %
39 %
39 %
39 %
39 %
39 %
39 %
40 %
40 %
40 %
40 %
40 %
40 %
40 %
41 %
41 %
41 %

41 %
41 %
42 %
42 %
42 %
42 %
42 %
42 %
42 %
43 %
43 %
43 %
43 %
43 %
43 %
44 %
44 %
44 %
44 %
44 %
44 %
45 %
45 %
45 %
45 %
45 %
45 %
45 %
46 %
46 %
46 %
46 %
46 %
46 %
47 %
47 %
47 %
47 %
47 %
47 %
47 %
48 %
48 %
48 %
48 %
48 %
48 %
49 %
49 %
49 %
49 %
49 %
49 %
50 %
50 %
50 %

50 %
50 %
50 %
51 %
51 %
51 %
51 %
51 %
51 %
52 %
52 %
52 %
52 %
52 %
52 %
52 %
53 %
53 %
53 %
53 %
53 %
53 %
54 %
54 %
54 %
54 %
54 %
54 %
55 %
55 %
55 %
55 %
55 %
55 %
55 %
56 %
56 %
56 %
56 %
56 %
56 %
57 %
57 %
57 %
57 %
57 %
57 %
57 %
57 %
58 %
58 %
58 %
58 %
58 %
58 %
59 %
59 %

```
59 %
59 %
59 %
60 %
60 %
60 %
60 %
60 %
60 %
60 %
61 %
61 %
61 %
61 %
61 %
61 %
62 %
62 %
62 %
62 %
62 %
62 %
62 %
63 %
63 %
63 %
63 %
63 %
63 %
64 %
64 %
64 %
64 %
64 %
64 %
65 %
65 %
65 %
65 %
65 %
65 %
65 %
66 %
66 %
66 %
66 %
66 %
66 %
67 %
67 %
67 %
67 %
67 %
67 %
67 %
68 %
```

68 %
68 %
68 %
68 %
69 %
69 %
69 %
69 %
69 %
70 %
70 %
70 %
70 %
70 %
70 %
71 %
71 %
71 %
71 %
71 %
72 %
72 %
72 %
72 %
72 %
72 %
73 %
73 %
73 %
73 %
73 %
74 %
74 %
74 %
74 %
74 %
75 %
75 %
75 %
75 %
75 %
75 %
76 %
76 %
76 %
76 %
76 %
77 %

77 %
77 %
77 %
77 %
77 %
78 %
78 %
78 %
78 %
78 %
79 %
79 %
79 %
79 %
79 %
80 %
80 %
80 %
80 %
80 %
80 %
81 %
81 %
81 %
81 %
81 %
82 %
82 %
82 %
82 %
82 %
82 %
83 %
83 %
83 %
83 %
83 %
84 %
84 %
84 %
84 %
84 %
85 %
85 %
85 %
85 %
85 %
85 %

86 %
86 %
86 %
86 %
86 %
87 %
87 %
87 %
87 %
87 %
87 %
88 %
88 %
88 %
88 %
88 %
89 %
89 %
89 %
89 %
89 %
90 %
90 %
90 %
90 %
90 %
90 %
91 %
91 %
91 %
91 %
91 %
92 %
92 %
92 %
92 %
92 %
92 %
93 %
93 %
93 %
93 %
93 %
94 %
94 %
94 %
94 %
94 %

```
95 %
95 %
95 %
95 %
95 %
95 %
96 %
96 %
96 %
96 %
96 %
96 %
97 %
97 %
97 %
97 %
97 %
97 %
97 %
98 %
98 %
98 %
98 %
98 %
98 %
99 %
99 %
99 %
99 %
99 %
-3573.3534202575684
```

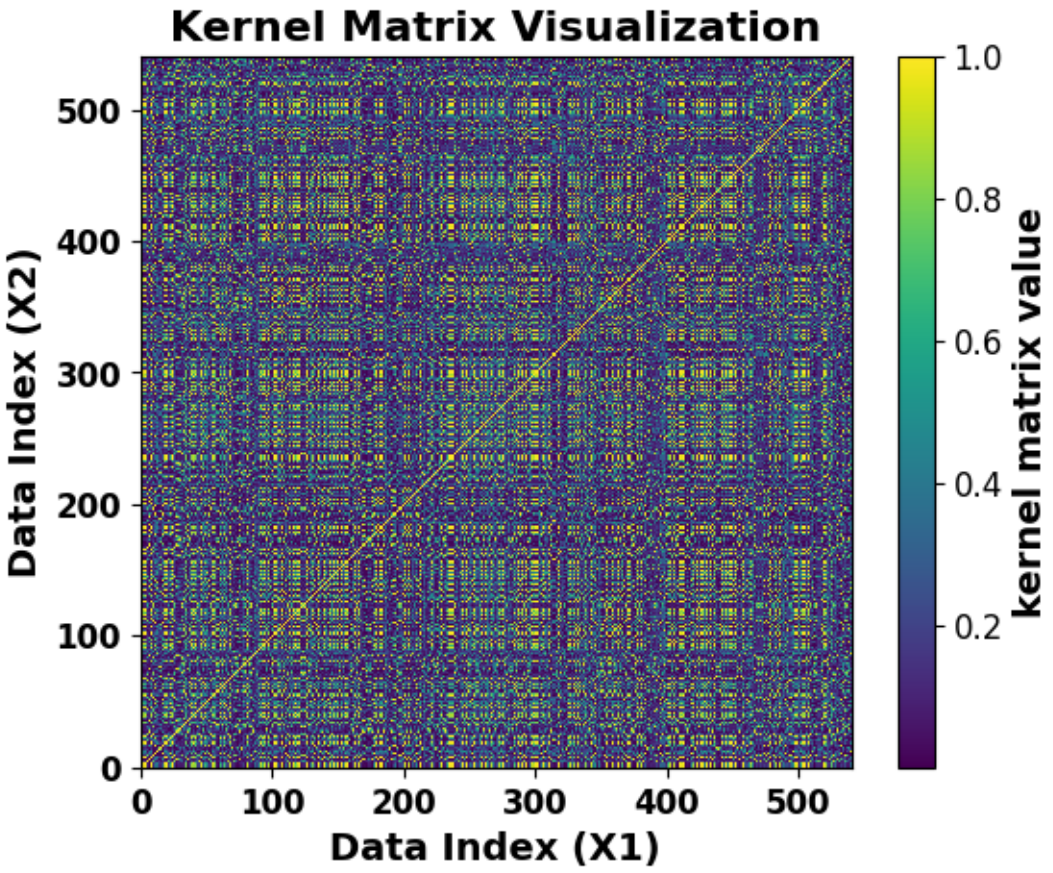
[ ]:
```
import numpy as np



# Save the matrix to a .npy file
np.save('./FEATURE_RESULTS/QKM_RESULTS_TWO_QUBITS/
 ↪simulation_TRAIN_kernel_matrix_DP_ruptura.npy', matrix_simetrica)
```

[ ]:
```
"""

This step is repeated for the other three combinations giving the results
within the directory /FEATURE_RESULTS .

With this simulation method
backend = Aer.get_backend('statevector_simulator')

simulation_TRAIN_kernel_matrix_DP_NODP.npy
simulation_TRAIN_kernel_matrix_DP_RAYOMAS.npy
```

```
simulation_TRAIN_kernel_matrix_DP_ruptura.npy
simulation_TRAIN_kernel_matrix_ruptura_NODP.npy


"""
```

[1]:
```
import numpy as np
matrix_simetrica=[]
# Supongamos que 'matrix' es la matriz del kernel que has calculado
#matrix = kernel_gram_matrix_full(x_train, x_train)

# load la matriz en un archivo .npy
matrix_simetrica=np.load('./FEATURE_RESULTS/QKM_RESULTS_TWO_QUBITS/
 ↪simulation_TRAIN_kernel_matrix_DP_ruptura.npy', matrix_simetrica)
```

[ ]:
```
import matplotlib.pyplot as plt

def visualize_kernel_matrix(kernel_matrix):
    plt.imshow(kernel_matrix, cmap='viridis', interpolation='none',␣
 ↪origin='lower', extent=[0, len(kernel_matrix), 0, len(kernel_matrix)])
    cbar = plt.colorbar(label='kernel matrix value')
    cbar.ax.yaxis.set_tick_params(labelsize=12)
    cbar.set_label('kernel matrix value', weight='bold', fontsize=14)
    plt.title('Kernel Matrix Visualization', fontweight='bold', fontsize=16)
    plt.xlabel('Data Index (X1)', fontweight='bold', fontsize=14)
    plt.ylabel('Data Index (X2)', fontweight='bold', fontsize=14)

    # Aumentar el tamaño y poner en negrita los números de los ejes
    plt.xticks(fontsize=12, fontweight='bold')
    plt.yticks(fontsize=12, fontweight='bold')

    plt.show()

# Supongamos que 'matrix' es la matriz del kernel que has calculado
# (asegúrate de que tiene valores entre 0 y 1)
# matrix = kernel_gram_matrix_full(x_train, x_train)

# Visualizar la matriz del kernel
visualize_kernel_matrix(matrix_simetrica)
```



[ ]:
```
matrix_simetrica[1,1]
```

[ ]:
```
0.9999999999999996
```

[ ]:
```
"""


Brief explanation of the method:

"This code uses SVM (Support Vector Machine) for classification, specifically␣
 ↪the implementation provided by scikit-learn in Python.

Here is a line-by-line explanation:

clf = svm.SVC(kernel="precomputed"): This creates an SVM classifier (SVC, which␣
 ↪stands for Support Vector Classifier) with a precomputed kernel.
The kernel parameter specifies the type of kernel to use in the SVM algorithm.
A precomputed kernel means that the classifier expects a precomputed kernel␣
 ↪matrix as input rather than the training data directly.
```

```
This can be useful in situations where kernel computation is expensive and has␣
 ↪already been performed outside the model training.

y_train = TRAIN_LABELS: This appears to be assigning the training labels to␣
 ↪y_train.
TRAIN_LABELS contains the labels corresponding to the training data used to␣
 ↪train the model.

clf.fit(matrix, y_train): This line is fitting the SVM model to the training␣
 ↪data.
The matrix contains the precomputed kernel matrix (or the training data if a␣
 ↪precomputed kernel is not used),
and y_train contains the corresponding labels for this data.
The fit() method adjusts the model to the provided data, which means it finds␣
 ↪the optimal hyperplane
that separates the different classes based on the training data and the␣
 ↪associated labels.
Once this process is complete, the model is ready to make predictions on new,␣
 ↪unseen data."

"""
```

[43]:
```
import numpy as np


# Other matrices obtained for the other combination  DP_ruptura by the same␣
 ↪procedure.
# Load matrix from .npy file
matrix_simetrica_cargada = np.load('./FEATURE_RESULTS/QKM_RESULTS_TWO_QUBITS/
 ↪simulation_TRAIN_kernel_matrix_DP_ruptura.npy')

# Print the matrix to verify
print(matrix_simetrica_cargada)
```

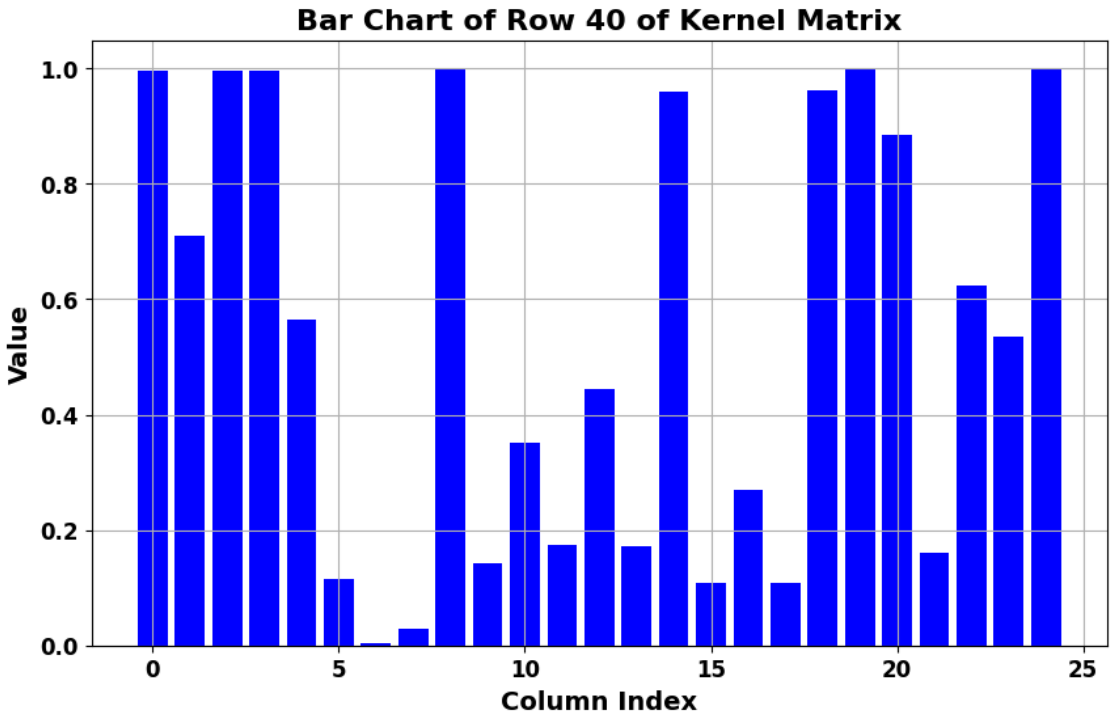
```
[[1.         0.71254132 0.99850877 … 0.06234615 0.83570996 0.33445261]
 [0.71254132 1.         0.69718794 … 0.24242795 0.54281082 0.1549732 ]
 [0.99850877 0.69718794 1.         … 0.07086102 0.84648396 0.34268151]
 …
 [0.06234615 0.24242795 0.07086102 … 1.         0.05001679 0.06052047]
 [0.83570996 0.54281082 0.84648396 … 0.05001679 1.         0.10556238]
 [0.33445261 0.1549732  0.34268151 … 0.06052047 0.10556238 1.        ]]
```

[47]:
```
import time
from sklearn import svm
from time import time
# Assuming XTest is your test dataset and YLabels are the corresponding labels
```

```
# It is assumed that you already have the TRAIN_LABELS and matrix variables␣
 ↪defined above

# Create an SVM classifier with the precomputed kernel


clf = svm.SVC(kernel="precomputed")

# Fit the model to the training data
"""
finds the optimal hyperplane that separates the different classes based on the␣
 ↪training data and the associated labels.
The objective function of the SVM is convex, meaning there are no local minima.
"""

start_time = time()
clf.fit(matrix_simetrica, TRAIN_LABELS)
training_time = time() - start_time
print("Tiempo de entrenamiento:", training_time, "segundos")
```

```
Tiempo de entrenamiento: 0.0036895275115966797 segundos
```

[3]:
```
import numpy as np
import matplotlib.pyplot as plt


# Suppose 'kernel_matrix' is the kernel matrix you computed
# with state vector simulator

# Select row 40 of the matrix
row_40 = matrix_simetrica[39]  # Index 39 why lists in Python are 0-indexed

# Crear el diagrama de barras de la fila 40
plt.figure(figsize=(10, 6))
plt.bar(range(0, 25), row_40[0:25], color='b')
plt.title('Bar Chart of Row 40 of Kernel Matrix', fontweight='bold',␣
 ↪fontsize=16)
plt.xlabel('Column Index', fontweight='bold', fontsize=14)
plt.ylabel('Value', fontweight='bold', fontsize=14)
plt.xticks(fontsize=12, fontweight='bold')
plt.yticks(fontsize=12, fontweight='bold')
plt.grid(True)
plt.show()
```



[ ]:
```
#Or, optionally use the save_account() method to save your credentials for easy␣
 ↪access later on, before initializing the service.
"""
This part is about estimating the kernel with a real quantum computer
those used are IBM Osaka;IBM Kyoto;IBM Brisbane.
Calibration data for computers used in the experiments from IBM. in  Table 2
del articulo:
Quantum Variational vs Quantum Kernel Machine Learning
Models for Partial Discharge Classification in Dielectric Oils
José Miguel Monzón-Verona , Santiago García-Alonso , and Francisco Jorge␣
 ↪Santana-Martín
"""
from qiskit_ibm_runtime import QiskitRuntimeService

# Save an IBM Quantum account and set it as your default account.

# Save an IBM Quantum account and set it as your default account.
QiskitRuntimeService.save_account(channel="ibm_quantum", token="........
 ↪8e5338f3752cf34eaa427.........", overwrite=True, set_as_default=True)
# Load saved credentials
service = QiskitRuntimeService()
backend = service.least_busy(operational=True, simulator=False)
backend.name
```

[ ]:
```
'ibm_kyoto'
```

[ ]:
```
from qiskit_ibm_runtime.fake_provider import␣
↪FakeManilaV2,FakeBrisbane,FakeKyoto,FakeOsaka
backend =FakeOsaka()
```

[ ]:
```
from qiskit.circuit import QuantumCircuit
from qiskit.transpiler.preset_passmanagers import generate_preset_pass_manager
from qiskit_ibm_runtime import SamplerV2 as Sampler
from qiskit_ibm_runtime.fake_provider import FakeManilaV2
```

[ ]:
```
from qiskit.circuit.library import ZZFeatureMap
zz_feature_map_reference = ZZFeatureMap(feature_dimension=2, reps=1)

zz_feature_map_reference.decompose().draw()
```

[ ]:
```
q_0:   H   P(2.0*x[0])

q_1:   H   P(2.0*x[1])   X   P(2.0*( - x[0])*( - x[1]))   X
```

[ ]:
```

```

[ ]:
```
"""
In this part, the functions used to estimate the kernel that is represented in␣
↪the
Figure 11. Generic structure of the quantum circuit and measurement used to␣
↪estimate the kernel of
Equation (26). It is particularized for three features.
"""

counts= []
circuits_a = []
circuits_aa = []

def scalar_product_circuitos(x, y):

    zz_feature_map_reference1 = zz_feature_map_reference.assign_parameters(y)

    zz_feature_map_reference_inv = zz_feature_map_reference.assign_parameters(x)

    zzinv1=zz_feature_map_reference_inv.inverse()
    # Combine the circuits
    circuito_final = zz_feature_map_reference1.compose(zzinv1, qubits=[0, 1])
    # Add measurements for each qubit
```

```
    circuito_final.measure_all()
    circuits_a.append(circuito_final)


    # backend = FakeOsaka()  for simulation
    passmanager = generate_preset_pass_manager(optimization_level=3,␣
 ↪backend=backend)

    transpiled_circuit = passmanager.run(circuito_final)
    circuits_aa.append(transpiled_circuit)


    return 1



def kernel_gram_matrix_circuitos(X1, X2):
    print("Calculando matriz de Gram")

    gram_matrix = np.zeros((X1.shape[0], X2.shape[0]))
    for i, x1 in enumerate(X1):
        print(int(i / len(X1) * 100), "%")
        for j, x2 in enumerate(X2):
            #qc = QuantumCircuit(3, 3)
            x1 = x1.flatten()
            x2 = x2.flatten()
            prob = scalar_product_circuitos(x1, x2)  # Use only the probability␣
 ↪of state '00'
            gram_matrix[i, j] = prob

    return gram_matrix
```

[ ]:
```
# Transpile circuit for noisy basis gates
passmanager = generate_preset_pass_manager(optimization_level=3,␣
 ↪backend=sim_noise)
circ_tnoise = passmanager.run(circ)
```

[ ]:
```
from qiskit_ibm_runtime import QiskitRuntimeService, SamplerV2 as Sampler

# Assuming you have already configured your Qiskit service and backend

# Initialize the sampler and run the circuits
sampler = Sampler(backend)
job = sampler.run(circuits_aa)
result = job.result()

# Iterate over the results to print and save the counts
```

```
with open('./FEATURE_RESULTS/QKM_RESULTS_TWO_QUBITS/
 ↪DP_ruptura_IBM_kyoto__fila_kernel39_25_results_counts.txt', 'w') as f:
    for idx, pub_result in enumerate(result):
        counts = pub_result.data.meas.get_counts()
        f.write(f"Counts for pub {idx}: {counts}\n")
        print(f" > Counts for pub {idx}: {counts}")
```

```
> Counts for pub 0: {'00': 4029, '01': 30, '11': 15, '10': 22}
> Counts for pub 1: {'00': 2903, '01': 753, '11': 128, '10': 312}
> Counts for pub 2: {'00': 4015, '10': 32, '01': 30, '11': 19}
> Counts for pub 3: {'00': 4007, '11': 24, '01': 29, '10': 36}
> Counts for pub 4: {'00': 2351, '01': 96, '11': 1524, '10': 125}
> Counts for pub 5: {'11': 1369, '00': 506, '01': 1765, '10': 456}
> Counts for pub 6: {'11': 2596, '01': 1436, '10': 36, '00': 28}
> Counts for pub 7: {'10': 3438, '00': 132, '01': 475, '11': 51}
> Counts for pub 8: {'00': 4014, '11': 13, '01': 41, '10': 28}
> Counts for pub 9: {'10': 1122, '01': 1807, '11': 550, '00': 617}
> Counts for pub 10: {'11': 2419, '00': 1310, '01': 207, '10': 160}
> Counts for pub 11: {'10': 2093, '01': 872, '00': 666, '11': 465}
> Counts for pub 12: {'11': 851, '01': 919, '00': 1769, '10': 557}
> Counts for pub 13: {'00': 680, '10': 1457, '11': 628, '01': 1331}
> Counts for pub 14: {'00': 3862, '11': 136, '01': 46, '10': 52}
> Counts for pub 15: {'10': 2793, '00': 475, '01': 670, '11': 158}
> Counts for pub 16: {'11': 2590, '00': 1069, '01': 308, '10': 129}
> Counts for pub 17: {'10': 2085, '00': 419, '01': 613, '11': 979}
> Counts for pub 18: {'00': 3849, '11': 144, '01': 52, '10': 51}
> Counts for pub 19: {'00': 4026, '10': 38, '01': 25, '11': 7}
> Counts for pub 20: {'00': 3587, '11': 406, '01': 28, '10': 75}
> Counts for pub 21: {'11': 2351, '01': 857, '00': 666, '10': 222}
> Counts for pub 22: {'00': 2547, '10': 120, '11': 1306, '01': 123}
> Counts for pub 23: {'11': 1765, '00': 2174, '10': 90, '01': 67}
> Counts for pub 24: {'00': 4018, '11': 11, '10': 35, '01': 32}
```

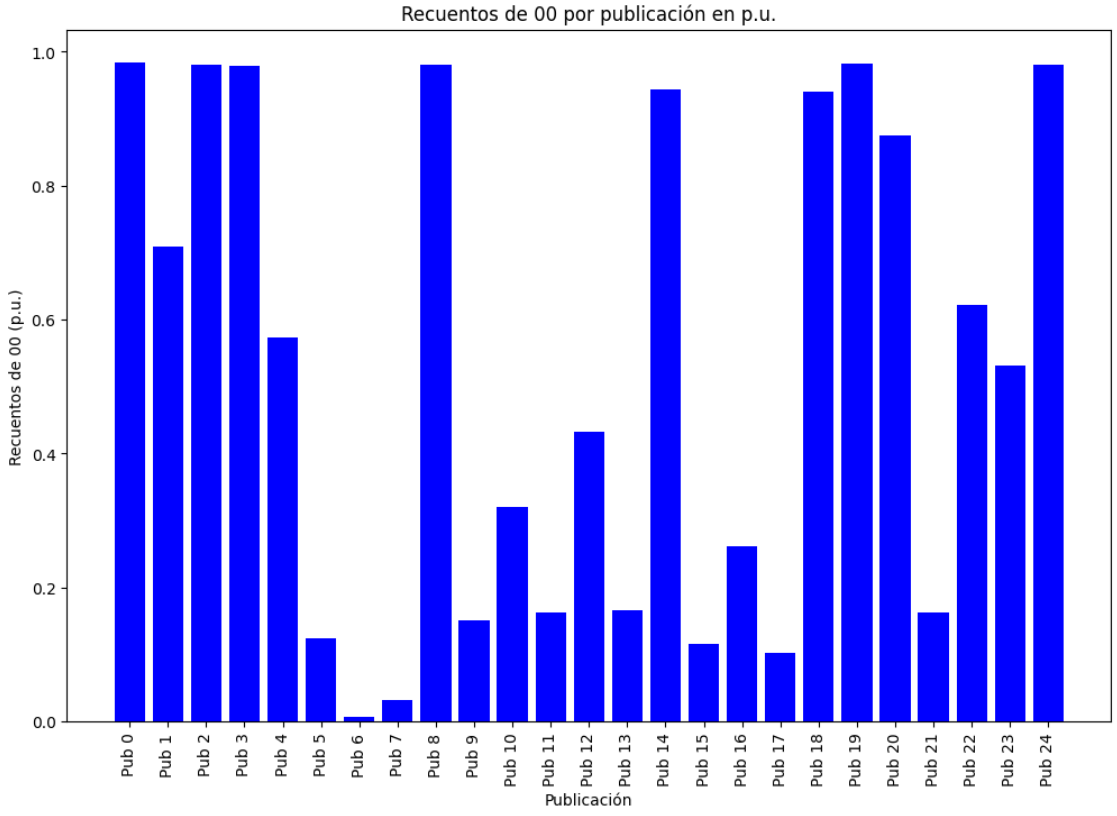
[ ]:
```
import matplotlib.pyplot as plt


#Assuming you already have the results in the `result` variable
counts_data = []
for idx, pub_result in enumerate(result):
    counts = pub_result.data.meas.get_counts()
    counts_data.append(counts)
    print(f" > Counts for pub {idx}: {counts}")


#Extract the counts for '00' from each post and calculate the total sum of␣
 ↪counts per post
counts_00 = []
```

```python
for counts in counts_data:
    total_counts = sum(counts.values())
    count_00 = counts.get('00', 0)
    count_00_pu = count_00 / total_counts if total_counts != 0 else 0  # Evitar␣
 ↪división por cero
    counts_00.append(count_00_pu)


#Print '00' counts in p.u. to verify
for idx, count in enumerate(counts_00):
    print(f"Counts of '00' in p.u. for pub {idx}: {count:.4f}")
```

```
 > Counts for pub 0: {'00': 4029, '01': 30, '11': 15, '10': 22}
 > Counts for pub 1: {'00': 2903, '01': 753, '11': 128, '10': 312}
 > Counts for pub 2: {'00': 4015, '10': 32, '01': 30, '11': 19}
 > Counts for pub 3: {'00': 4007, '11': 24, '01': 29, '10': 36}
 > Counts for pub 4: {'00': 2351, '01': 96, '11': 1524, '10': 125}
 > Counts for pub 5: {'11': 1369, '00': 506, '01': 1765, '10': 456}
 > Counts for pub 6: {'11': 2596, '01': 1436, '10': 36, '00': 28}
 > Counts for pub 7: {'10': 3438, '00': 132, '01': 475, '11': 51}
 > Counts for pub 8: {'00': 4014, '11': 13, '01': 41, '10': 28}
 > Counts for pub 9: {'10': 1122, '01': 1807, '11': 550, '00': 617}
 > Counts for pub 10: {'11': 2419, '00': 1310, '01': 207, '10': 160}
 > Counts for pub 11: {'10': 2093, '01': 872, '00': 666, '11': 465}
 > Counts for pub 12: {'11': 851, '01': 919, '00': 1769, '10': 557}
 > Counts for pub 13: {'00': 680, '10': 1457, '11': 628, '01': 1331}
 > Counts for pub 14: {'00': 3862, '11': 136, '01': 46, '10': 52}
 > Counts for pub 15: {'10': 2793, '00': 475, '01': 670, '11': 158}
 > Counts for pub 16: {'11': 2590, '00': 1069, '01': 308, '10': 129}
 > Counts for pub 17: {'10': 2085, '00': 419, '01': 613, '11': 979}
 > Counts for pub 18: {'00': 3849, '11': 144, '01': 52, '10': 51}
 > Counts for pub 19: {'00': 4026, '10': 38, '01': 25, '11': 7}
 > Counts for pub 20: {'00': 3587, '11': 406, '01': 28, '10': 75}
 > Counts for pub 21: {'11': 2351, '01': 857, '00': 666, '10': 222}
 > Counts for pub 22: {'00': 2547, '10': 120, '11': 1306, '01': 123}
 > Counts for pub 23: {'11': 1765, '00': 2174, '10': 90, '01': 67}
 > Counts for pub 24: {'00': 4018, '11': 11, '10': 35, '01': 32}
Counts of '00' in p.u. for pub 0: 0.9836
Counts of '00' in p.u. for pub 1: 0.7087
Counts of '00' in p.u. for pub 2: 0.9802
Counts of '00' in p.u. for pub 3: 0.9783
Counts of '00' in p.u. for pub 4: 0.5740
Counts of '00' in p.u. for pub 5: 0.1235
Counts of '00' in p.u. for pub 6: 0.0068
Counts of '00' in p.u. for pub 7: 0.0322
Counts of '00' in p.u. for pub 8: 0.9800
Counts of '00' in p.u. for pub 9: 0.1506
```

```
Counts of '00' in p.u. for pub 10: 0.3198
Counts of '00' in p.u. for pub 11: 0.1626
Counts of '00' in p.u. for pub 12: 0.4319
Counts of '00' in p.u. for pub 13: 0.1660
Counts of '00' in p.u. for pub 14: 0.9429
Counts of '00' in p.u. for pub 15: 0.1160
Counts of '00' in p.u. for pub 16: 0.2610
Counts of '00' in p.u. for pub 17: 0.1023
Counts of '00' in p.u. for pub 18: 0.9397
Counts of '00' in p.u. for pub 19: 0.9829
Counts of '00' in p.u. for pub 20: 0.8757
Counts of '00' in p.u. for pub 21: 0.1626
Counts of '00' in p.u. for pub 22: 0.6218
Counts of '00' in p.u. for pub 23: 0.5308
Counts of '00' in p.u. for pub 24: 0.9810
```

[ ]:
```python
# Create the bar chart
fig, ax = plt.subplots(figsize=(12, 8))

# Define the indexes and values for the chart
pub_indices = range(len(counts_00))

# Create the bars
ax.bar(pub_indices, counts_00, color='blue')

# Add tags and title
ax.set_xlabel('Publicación')
ax.set_ylabel('Recuentos de 00 (p.u.)')
ax.set_title('Recuentos de 00 por publicación en p.u.')
ax.set_xticks(pub_indices)
ax.set_xticklabels([f'Pub {i}' for i in pub_indices], rotation=90)

# Mostrar el gráfico
plt.show()
```

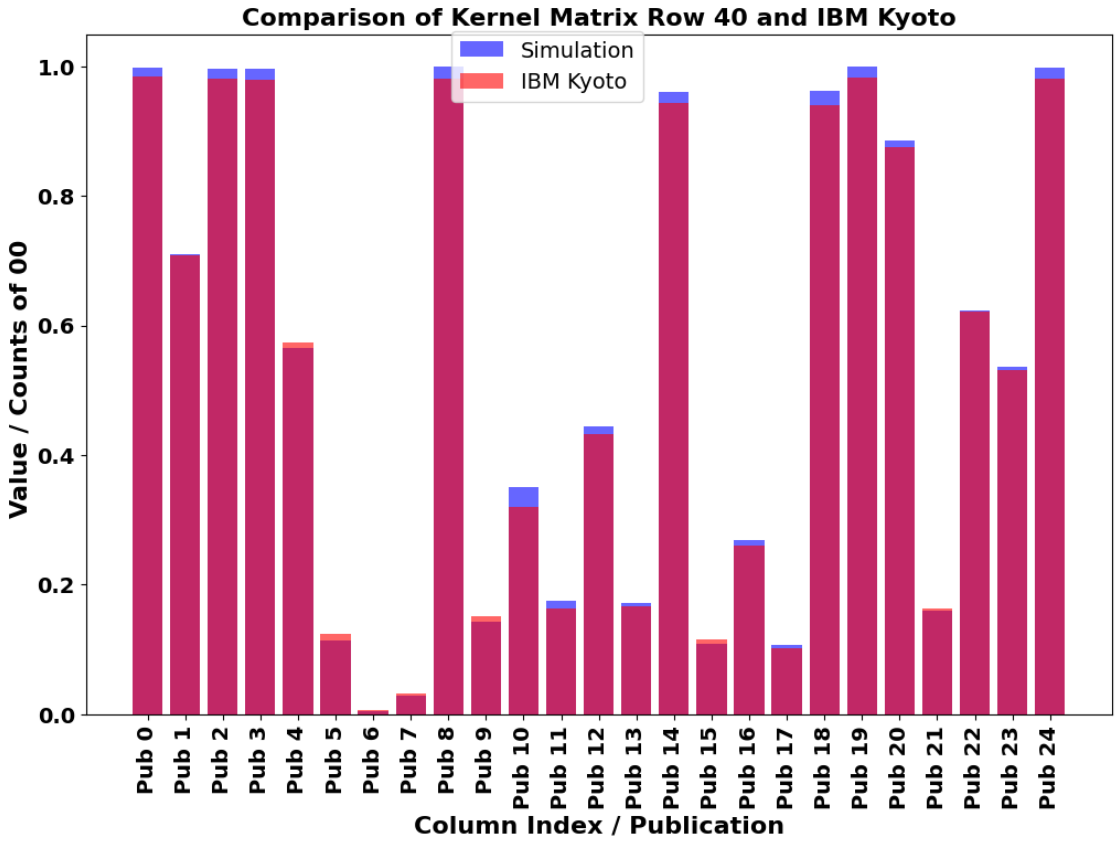


[ ]:
```python
import numpy as np
import matplotlib.pyplot as plt
import ast

# Path to the saved counts data file
file_path = './FEATURE_RESULTS/QKM_RESULTS_TWO_QUBITS/
 ↪DP_ruptura_IBM_Kyoto__fila_kernel39_25_results_counts.txt'

# Initialize an empty list to hold the counts data
counts_data = []

# Read the file and extract counts for each publication
with open(file_path, 'r') as f:
    for line in f:
        if line.startswith('Counts for pub'):
            parts = line.split(': ', 1)  # Split only at the first occurrence
            try:
                counts = ast.literal_eval(parts[1].strip())
                counts_data.append(counts)
            except (SyntaxError, ValueError) as e:
                print(f"Error parsing line: {line.strip()}")
```

```
                print(e)

# Extract the counts for '00' from each publication and normalize them
counts_00 = []
for counts in counts_data:
    total_counts = sum(counts.values())
    counts_00.append(counts.get('00', 0) / total_counts if total_counts > 0␣
 ↪else 0)

# Print the normalized counts for '00' to verify
for idx, count in enumerate(counts_00):
    print(f"Normalized counts of '00' for pub {idx}: {count:.4f}")

# Assume 'matrix_simetrica' is the kernel matrix you have loaded


# Select row 40 from the matrix
row_40 = matrix_simetrica[39]  # Index 39 because Python lists are 0-indexed

# Ensure both lists have the same length
length = min(len(row_40[0:25]), len(counts_00))
row_40 = row_40[0:length]
counts_00 = counts_00[:length]

# Create the combined bar chart
fig, ax = plt.subplots(figsize=(12, 8))

# Define the indices for the plot
indices = range(length)

# Create the bars for row 40 of the kernel matrix
ax.bar(indices, row_40, color='b', alpha=0.6, label='Simulation')

# Create the bars for the normalized counts of 00
ax.bar(indices, counts_00, color='r', alpha=0.6, label='IBM Kyoto')

# Add labels and title in bold and increase font size
ax.set_xlabel('Column Index / Publication', fontweight='bold', fontsize=16)
ax.set_ylabel('Value / Counts of 00 ', fontweight='bold', fontsize=16)
ax.set_title('Comparison of Kernel Matrix Row 40 and IBM Kyoto',␣
 ↪fontweight='bold', fontsize=16)

# Modify the x-axis tick labels
ax.set_xticks(indices)
ax.set_xticklabels([f'Pub {i}' for i in indices], rotation=90, fontsize=14,␣
 ↪fontweight='bold')
```

```
# Modify the y-axis tick labels
plt.yticks(fontsize=14, fontweight='bold')


ax.legend(loc='upper center', bbox_to_anchor=(0.45, 1.02), fontsize=14)
# Show the plot
plt.show()
```

```
Normalized counts of '00' for pub 0: 0.9836
Normalized counts of '00' for pub 1: 0.7087
Normalized counts of '00' for pub 2: 0.9802
Normalized counts of '00' for pub 3: 0.9783
Normalized counts of '00' for pub 4: 0.5740
Normalized counts of '00' for pub 5: 0.1235
Normalized counts of '00' for pub 6: 0.0068
Normalized counts of '00' for pub 7: 0.0322
Normalized counts of '00' for pub 8: 0.9800
Normalized counts of '00' for pub 9: 0.1506
Normalized counts of '00' for pub 10: 0.3198
Normalized counts of '00' for pub 11: 0.1626
Normalized counts of '00' for pub 12: 0.4319
Normalized counts of '00' for pub 13: 0.1660
Normalized counts of '00' for pub 14: 0.9429
Normalized counts of '00' for pub 15: 0.1160
Normalized counts of '00' for pub 16: 0.2610
Normalized counts of '00' for pub 17: 0.1023
Normalized counts of '00' for pub 18: 0.9397
Normalized counts of '00' for pub 19: 0.9829
Normalized counts of '00' for pub 20: 0.8757
Normalized counts of '00' for pub 21: 0.1626
Normalized counts of '00' for pub 22: 0.6218
Normalized counts of '00' for pub 23: 0.5308
Normalized counts of '00' for pub 24: 0.9810
```



[ ]:
```
backend.name
```

[ ]:
```
'ibm_kyoto'
```

[ ]:
```
print(countss)
```

```
Result(backend_name='aer_simulator', backend_version='0.14.1', qobj_id='',
job_id='d4a4144a-a806-4293-8621-97d1a3532a94', success=True,
results=[ExperimentResult(shots=100, success=True, meas_level=2,
data=ExperimentResultData(counts={'0x2': 2, '0x0': 98}),
header=QobjExperimentHeader(creg_sizes=[['meas', 2]],
global_phase=6.283185307179585, memory_slots=2, n_qubits=127,
name='circuit-469644', qreg_sizes=[['q', 127]], metadata={}), status=DONE,
seed_simulator=2408177570, metadata={'time_taken': 0.000516568,
'num_bind_params': 1, 'parallel_state_update': 2, 'parallel_shots': 1,
'required_memory_mb': 1, 'input_qubit_map': [[29, 1], [28, 0]], 'method':
'density_matrix', 'device': 'CPU', 'num_qubits': 2, 'sample_measure_time':
4.4479e-05, 'active_input_qubits': [28, 29], 'num_clbits': 2, 'remapped_qubits':
True, 'runtime_parameter_bind': False, 'max_memory_mb': 12978, 'noise':
'superop', 'measure_sampling': True, 'batched_shots_optimization': False,
'fusion': {'applied': False, 'max_fused_qubits': 2, 'threshold': 7, 'enabled':
```

```
True}}, time_taken=0.000516568), ExperimentResult(shots=100, success=True,
meas_level=2, data=ExperimentResultData(counts={'0x3': 3, '0x2': 4, '0x0': 76,
'0x1': 17}), header=QobjExperimentHeader(creg_sizes=[['meas', 2]],
global_phase=1.589995892154473, memory_slots=2, n_qubits=127,
name='circuit-469666', qreg_sizes=[['q', 127]], metadata={}), status=DONE,
seed_simulator=2408179683, metadata={'time_taken': 0.001090596,
'num_bind_params': 1, 'parallel_state_update': 2, 'parallel_shots': 1,
'required_memory_mb': 1, 'input_qubit_map': [[29, 1], [28, 0]], 'method':
'density_matrix', 'device': 'CPU', 'num_qubits': 2, 'sample_measure_time':
3.1659e-05, 'active_input_qubits': [28, 29], 'num_clbits': 2, 'remapped_qubits':
True, 'runtime_parameter_bind': False, 'max_memory_mb': 12978, 'noise':
'superop', 'measure_sampling': True, 'batched_shots_optimization': False,
'fusion': {'applied': False, 'max_fused_qubits': 2, 'threshold': 7, 'enabled':
True}}, time_taken=0.001090596), ExperimentResult(shots=100, success=True,
meas_level=2, data=ExperimentResultData(counts={'0x3': 7, '0x1': 21, '0x2': 5,
'0x0': 67}), header=QobjExperimentHeader(creg_sizes=[['meas', 2]],
global_phase=4.69318941502511, memory_slots=2, n_qubits=127,
name='circuit-469688', qreg_sizes=[['q', 127]], metadata={}), status=DONE,
seed_simulator=2408181796, metadata={'time_taken': 0.000802026,
'num_bind_params': 1, 'parallel_state_update': 2, 'parallel_shots': 1,
'required_memory_mb': 1, 'input_qubit_map': [[29, 1], [28, 0]], 'method':
'density_matrix', 'device': 'CPU', 'num_qubits': 2, 'sample_measure_time':
2.6076e-05, 'active_input_qubits': [28, 29], 'num_clbits': 2, 'remapped_qubits':
True, 'runtime_parameter_bind': False, 'max_memory_mb': 12978, 'noise':
'superop', 'measure_sampling': True, 'batched_shots_optimization': False,
'fusion': {'applied': False, 'max_fused_qubits': 2, 'threshold': 7, 'enabled':
True}}, time_taken=0.000802026), ExperimentResult(shots=100, success=True,
meas_level=2, data=ExperimentResultData(counts={'0x0': 100}),
header=QobjExperimentHeader(creg_sizes=[['meas', 2]],
global_phase=4.440892098500626e-16, memory_slots=2, n_qubits=127,
name='circuit-469710', qreg_sizes=[['q', 127]], metadata={}), status=DONE,
seed_simulator=2408183909, metadata={'time_taken': 0.000376095,
'num_bind_params': 1, 'parallel_state_update': 2, 'parallel_shots': 1,
'required_memory_mb': 1, 'input_qubit_map': [[29, 1], [28, 0]], 'method':
'density_matrix', 'device': 'CPU', 'num_qubits': 2, 'sample_measure_time':
2.2098e-05, 'active_input_qubits': [28, 29], 'num_clbits': 2, 'remapped_qubits':
True, 'runtime_parameter_bind': False, 'max_memory_mb': 12978, 'noise':
'superop', 'measure_sampling': True, 'batched_shots_optimization': False,
'fusion': {'applied': False, 'max_fused_qubits': 2, 'threshold': 7, 'enabled':
True}}, time_taken=0.000376095)], date=2024-05-25T18:05:11.865634,
status=COMPLETED, header=None, metadata={'time_taken_parameter_binding':
0.000323837, 'time_taken_execute': 0.164118279, 'omp_enabled': True,
'max_gpu_memory_mb': 0, 'max_memory_mb': 12978, 'parallel_experiments': 1},
time_taken=2.000185251235962)
```

[3]:
```
print("Checking SVC with train...")
x_train=TRAIN_DATA
```

```
from sklearn import svm
import numpy as np
from time import time
# Load matrix from .npy file
K = np.load('./FEATURE_RESULTS/QKM_RESULTS_TWO_QUBITS/
 ↪simulation_TRAIN_kernel_matrix_DP_ruptura.npy')
#K = np.load('./FEATURE_RESULTS/simulation_TRAIN_kernel_matrix_DP_NODP.npy')

clf = svm.SVC(kernel="precomputed")
clf.fit(K, TRAIN_LABELS)

#Record the start time
inicio_tiempo = time()
sol = clf.predict(K)

y_train=TRAIN_LABELS
success = 0


# Calculate the time difference
tiempo_ejecucion = time()- inicio_tiempo

print(f"Tiempo de ejecución: {tiempo_ejecucion} segundos")
```

```
Checking SVC with train…
Tiempo de ejecución: 0.002384185791015625 segundos
```

[46]:
```
inicio_tiempo = time()
y_train=TRAIN_LABELS
success = 0
for i in range(len(y_train)):
    if sol[i] == y_train[i]:
        success += 1

print("Precisión del train: ", success/len(sol)*100, "%")

# Registra el tiempo de finalización
fin_tiempo = time()

# Calcula la diferencia de tiempo
tiempo_ejecucion = fin_tiempo - inicio_tiempo

print(f"Tiempo de ejecución de compro train: {tiempo_ejecucion} segundos")
```

```
Precisión del train:  90.0 %
Tiempo de ejecución de compro train: 0.00032782554626464844 segundos
```

[ ]:
```
print("Comprobando con test...")
x_test=TEST_DATA
# Registra el tiempo de inicio
inicio_tiempo = time.time()
sol = clf.predict(kernel_gram_matrix_full(x_test, x_train))

y_test=TEST_LABELS
success = 0


# Calcula la diferencia de tiempo
tiempo_ejecucion = time.time()- inicio_tiempo

print(f"Tiempo de ejecución: {tiempo_ejecucion} segundos")
```

```
Comprobando con test…
Calculando matriz de Gram
0 %
0 %
1 %
2 %
2 %
3 %
4 %
5 %
5 %
6 %
7 %
8 %
8 %
9 %
10 %
11 %
11 %
12 %
13 %
13 %
14 %
15 %
16 %
16 %
17 %
18 %
19 %
19 %
20 %
21 %
```

22 %
22 %
23 %
24 %
25 %
25 %
26 %
27 %
27 %
28 %
29 %
30 %
30 %
31 %
32 %
33 %
33 %
34 %
35 %
36 %
36 %
37 %
38 %
38 %
39 %
40 %
41 %
41 %
42 %
43 %
44 %
44 %
45 %
46 %
47 %
47 %
48 %
49 %
50 %
50 %
51 %
52 %
52 %
53 %
54 %
55 %
55 %
56 %

57 %
58 %
58 %
59 %
60 %
61 %
61 %
62 %
63 %
63 %
64 %
65 %
66 %
66 %
67 %
68 %
69 %
69 %
70 %
71 %
72 %
72 %
73 %
74 %
75 %
75 %
76 %
77 %
77 %
78 %
79 %
80 %
80 %
81 %
82 %
83 %
83 %
84 %
85 %
86 %
86 %
87 %
88 %
88 %
89 %
90 %
91 %
91 %

```
92 %
93 %
94 %
94 %
95 %
96 %
97 %
97 %
98 %
99 %
Tiempo de ejecución: 1768.2468783855438 segundos
```

[ ]:
```python
y_test=TEST_LABELS
success = 0
for i in range(len(y_test)):
    if sol[i] == y_test[i]:
        success += 1

print("Precisión del train: ", success/len(sol)*100, "%")

# Registra el tiempo de finalización
fin_tiempo = time.time()

# Calcula la diferencia de tiempo
tiempo_ejecucion = fin_tiempo - inicio_tiempo

print(f"Tiempo de ejecución de compro test: {tiempo_ejecucion} segundos")
```

```
Precisión del train:  90.44117647058823 %
Tiempo de ejecución de compro test: 1827.001841545105 segundos
```

[ ]:
```python
""" Reproducibility  Support vector machine, SVM  with quantum kernel␣
 ↪estimation for 3 qubits

Table 5. Accuracy and execution times for three qubits Equation (28), with␣
 ↪symmetric matrix,
test_size=80%.




"""
```

[16]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle

import warnings
warnings.filterwarnings("ignore")
# constants
n = 2
RANDOM_STATE = 42
LR = 1e-3
class_labels = ['0', '1']

def normalizeData(DATA_PATH = "./FEATURE_RESULTS/FEATURE_resultante_DP_RAYOMAS.
 ↪csv"):
    """
    Normalizes the data
    """
    # Reads the data for three qubits
    data = pd.read_csv(DATA_PATH)
    data = shuffle(data, random_state=RANDOM_STATE)
    X, Y = data[['area_pixels', ' mean_coords_x',' mean_coords_y']].values,␣
 ↪data[' class'].values

    # normalize the data
    scaler = MinMaxScaler(feature_range=(-0 * np.pi, 2 * np.pi))
    X = scaler.fit_transform(X)
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.8,␣
 ↪random_state=RANDOM_STATE)
    return X_train, X_test, Y_train, Y_test


# In[5]:


import qiskit_algorithms
from qiskit_algorithms.optimizers import SPSA

from qiskit import QuantumCircuit

from qiskit.circuit.library import ZZFeatureMap, RealAmplitudes
from qiskit.quantum_info import Statevector

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler
```

```
from sklearn.model_selection import train_test_split

from sklearn.utils import shuffle

import warnings
warnings.filterwarnings("ignore")

TRAIN_DATA, TEST_DATA, TRAIN_LABELS, TEST_LABELS = normalizeData()
# Replace all occurrences of 2 with 1
TRAIN_LABELS = np.where(TRAIN_LABELS == 2, 1, TRAIN_LABELS)
TEST_LABELS = np.where(TEST_LABELS == 2, 1, TEST_LABELS)
```

[27]:
```
print(TRAIN_LABELS)
print(f"Dimensiones de TRAIN_LABELS: {TRAIN_LABELS.shape}")
```

```
[0 1 0 1 1 1 0 1 0 1 0 1 1 1 0 0 0 1 0 1 1 1 0 1 0 1 0 1 1 0 1 1 0 0 1 0 0
 1 1 1 0 0 1 1 1 0 0 0 1 1 0 0 1 0 0 0 0 0 0 1 1 0 0 1 1 0 0 1 0 0 1 0 0 1
 1 1 1 0 0 0 1 1 0 0 0 0 0 1 0 1 1 1 1 1 1 1 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0
 1 1 0 0 0 1 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 1 0 1 0 1 0 0 1 0 0 1
 0 0 0 0 1 1 1 0 1 1 1 0 1 0 0 1 1 0 0 1 0 0 0 0 1]
Dimensiones de TRAIN_LABELS: (173,)
```

[45]:
```
# FIRST METHOD FOR CALCULATING THE KERNEL WITH STATE VECTOR
import numpy as np
from qiskit import QuantumCircuit, transpile, assemble
import matplotlib.pyplot as plt
from qiskit_aer import AerSimulator
from qiskit.quantum_info import Statevector
```

[44]:
```
from qiskit.circuit import QuantumCircuit
from qiskit.transpiler.preset_passmanagers import generate_preset_pass_manager
from qiskit_ibm_runtime import SamplerV2 as Sampler
from qiskit_ibm_runtime.fake_provider import FakeManilaV2
from qiskit.circuit.library import ZZFeatureMap
zz_feature_map_reference = ZZFeatureMap(feature_dimension=3, reps=1)# 3 qubits
```

[17]:
```
counts= []
circuits_a = []
circuits_aa = []
#for multiple circuits
def scalar_product_circuitos(x, y):

    zz_feature_map_reference1 = zz_feature_map_reference.assign_parameters(y)

    zz_feature_map_reference_inv = zz_feature_map_reference.assign_parameters(x)

    zzinv1=zz_feature_map_reference_inv.inverse()
```

```
    # Combine the circuits
    circuito_final = zz_feature_map_reference1.compose(zzinv1, qubits=[0, 1,2])

    circuito_final.measure_all()
    circuits_a.append(circuito_final)

    """
    backend = Aer.get_backend('statevector_simulator')
    t_qc = transpile(circuito_final, backend)
    qobj = assemble(t_qc)
    result = backend.run(qobj).result()
    statevector = result.get_statevector(circuito_final)
    probs = np.abs(statevector)**2


"""
    #backend = FakeOsaka()
    passmanager = generate_preset_pass_manager(optimization_level=3,␣
 ↪backend=backend)
    #circ_tnoise = passmanager.run(circ)
    #shots=52000
    transpiled_circuit = passmanager.run(circuito_final)
    circuits_aa.append(transpiled_circuit)
    return 1
```

[ ]:

[47]:
```
zz_feature_map_reference = ZZFeatureMap(feature_dimension=3, reps=1)# 3 qubits␣
 ↪or n_qubits



def scalar_product_N_q(circuit, x, y):


    zz_feature_map_reference1 = zz_feature_map_reference.assign_parameters(y)

    zz_feature_map_reference_inv = zz_feature_map_reference.assign_parameters(x)

    zzinv1=zz_feature_map_reference_inv.inverse()
    #  Combine the circuits
    circuito_final = zz_feature_map_reference1.compose(zzinv1, qubits=[0, 1,2])␣
 ↪ # for 3 qubits



    backend = Aer.get_backend('statevector_simulator')
```

```
    t_qc = transpile(circuito_final, backend)
    qobj = assemble(t_qc)
    result = backend.run(qobj).result()
    statevector = result.get_statevector(t_qc)
    probs = np.abs(statevector)**2
    return probs




def kernel_gram_matrix_full(X1, X2):
    print("Calculando matriz de Gram")

    gram_matrix = np.zeros((X1.shape[0], X2.shape[0]))
    for i, x1 in enumerate(X1):
        print(int(i / len(X1) * 100), "%")
        for j, x2 in enumerate(X2):
            qc = QuantumCircuit(3, 3)
            x1 = x1.flatten()
            x2 = x2.flatten()
            prob = scalar_product_N_q( qc,x1, x2)[0]  # Usar solo la␣
 ↪probabilidad del estado '00'
            gram_matrix[i, j] = prob

    return gram_matrix







import numpy as np
import time
from qiskit import QuantumCircuit




def kernel_gram_matrix_full_symmetric(X1, X2):
    print("Calculando matriz de Gram")

    num_samples_X1 = X1.shape[0]
```

```
    gram_matrix = np.zeros((num_samples_X1, num_samples_X1))  # Make sure you␣
 ↪have the correct size

    # Calculate only the upper half of the matrix
    for i in range(num_samples_X1):
        print(int(i / num_samples_X1 * 100), "%")
        for j in range(i, num_samples_X1):  # Only calculate the top half
            qc = QuantumCircuit(3, 3)
            x1 = X1[i].flatten()
            x2 = X2[j].flatten()

            prob = scalar_product_N_q(qc, x1, x2)[0]
            gram_matrix[i, j] = prob
            gram_matrix[j, i] = prob  # Symmetry: assigning the same value to␣
 ↪the lower half

    return gram_matrix
x_train=TRAIN_DATA[:]
import time
inicio_tiempo = time.time()
matrix_simetrica = kernel_gram_matrix_full_symmetric(x_train, x_train)
fin=time.time()
tiempo=inicio_tiempo-fin
print(tiempo)
```

```
Calculando matriz de Gram
0 %
0 %
1 %
1 %
2 %
2 %
3 %
4 %
4 %
5 %
5 %
6 %
6 %
7 %
8 %
8 %
9 %
9 %
10 %
```

10 %
11 %
12 %
12 %
13 %
13 %
14 %
15 %
15 %
16 %
16 %
17 %
17 %
18 %
19 %
19 %
20 %
20 %
21 %
21 %
22 %
23 %
23 %
24 %
24 %
25 %
26 %
26 %
27 %
27 %
28 %
28 %
29 %
30 %
30 %
31 %
31 %
32 %
32 %
33 %
34 %
34 %
35 %
35 %
36 %
36 %
37 %
38 %

38 %
39 %
39 %
40 %
41 %
41 %
42 %
42 %
43 %
43 %
44 %
45 %
45 %
46 %
46 %
47 %
47 %
48 %
49 %
49 %
50 %
50 %
51 %
52 %
52 %
53 %
53 %
54 %
54 %
55 %
56 %
56 %
57 %
57 %
58 %
58 %
59 %
60 %
60 %
61 %
61 %
62 %
63 %
63 %
64 %
64 %
65 %
65 %

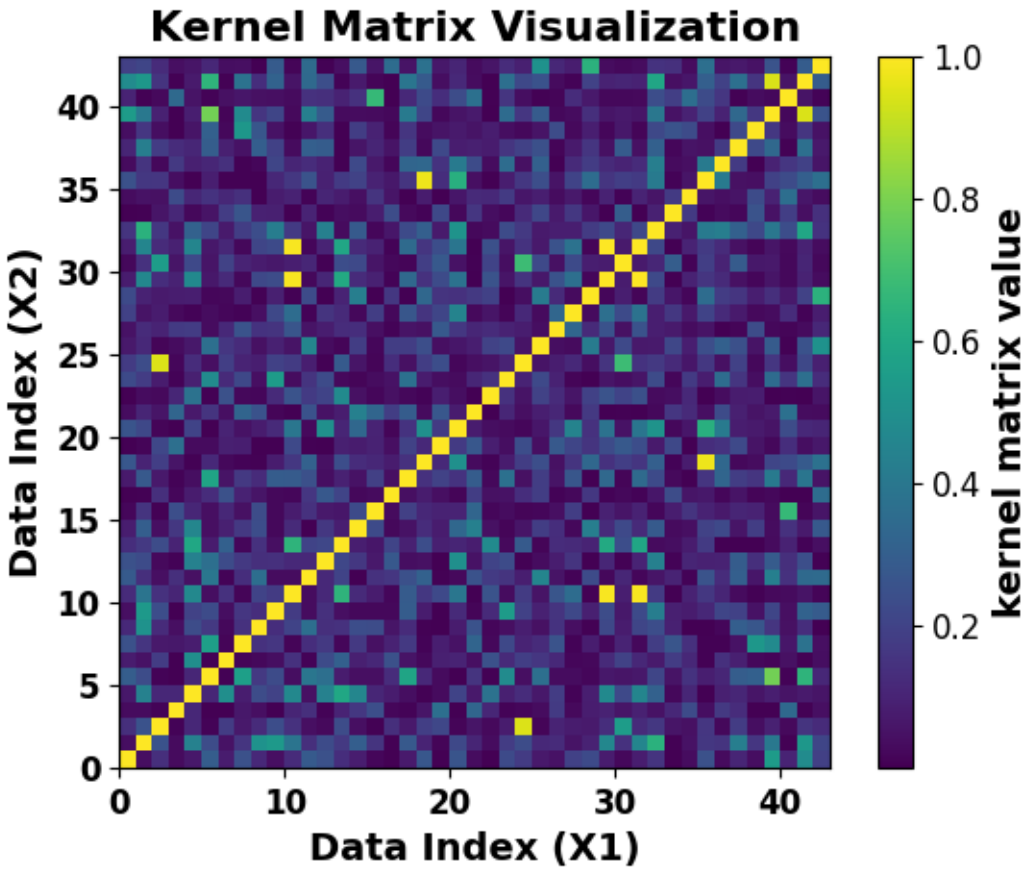
66 %
67 %
67 %
68 %
68 %
69 %
69 %
70 %
71 %
71 %
72 %
72 %
73 %
73 %
74 %
75 %
75 %
76 %
76 %
77 %
78 %
78 %
79 %
79 %
80 %
80 %
81 %
82 %
82 %
83 %
83 %
84 %
84 %
85 %
86 %
86 %
87 %
87 %
88 %
89 %
89 %
90 %
90 %
91 %
91 %
92 %
93 %
93 %

```
94 %
94 %
95 %
95 %
96 %
97 %
97 %
98 %
98 %
99 %
-215.4209179878235
```

[48]:
```python
import numpy as np



# Guardar la matriz en un archivo .npy
np.save('./FEATURE_RESULTS/QKM_RESULTS_THREE_QUBITS/
 ↪simulation_tres_q_TRAIN_kernel_matrix_DP_rayomas.npy', matrix_simetrica)
```

[24]:
```python
# load la matriz en un archivo .npy
matrix_simetrica_load=np.load('./FEATURE_RESULTS/QKM_RESULTS_THREE_QUBITS/
 ↪simulation_tres_q_TRAIN_kernel_matrix_DP_rayomas.npy')
```

[49]:
```python
import matplotlib.pyplot as plt

def visualize_kernel_matrix(kernel_matrix):
    plt.imshow(kernel_matrix, cmap='viridis', interpolation='none',␣
 ↪origin='lower', extent=[0, len(kernel_matrix), 0, len(kernel_matrix)])
    cbar = plt.colorbar(label='kernel matrix value')
    cbar.ax.yaxis.set_tick_params(labelsize=12)
    cbar.set_label('kernel matrix value', weight='bold', fontsize=14)
    plt.title('Kernel Matrix Visualization', fontweight='bold', fontsize=16)
    plt.xlabel('Data Index (X1)', fontweight='bold', fontsize=14)
    plt.ylabel('Data Index (X2)', fontweight='bold', fontsize=14)

    # Aumentar el tamaño y poner en negrita los números de los ejes
    plt.xticks(fontsize=12, fontweight='bold')
    plt.yticks(fontsize=12, fontweight='bold')

    plt.show()



#  Viewing the kernel matrix
visualize_kernel_matrix(matrix_simetrica_load)
```



[50]:
```
import time
from sklearn import svm


# Assuming XTest is your test dataset and YLabels are the corresponding labels
# It is assumed that you already have the TRAIN_LABELS and matrix variables␣
 ↪defined above


#Create a SVM classifier with the precomputed kernel
clf = svm.SVC(kernel="precomputed")


#Fitting the model to the training data
"""


Finds the optimal hyperplane that separates different classes based on the␣
 ↪training data and associated labels.
```

```
The SVM objective function is convex, meaning there are no local minima.


"""
start_time = time.time()
clf.fit(matrix_simetrica, TRAIN_LABELS)
training_time = time.time() - start_time
print("Training time:", training_time, "s")
```

Tiempo de entrenamiento: 0.0033407211303710938 segundos

[51]:
```
#Record the start time
inicio_tiempo = time.time()
sol = clf.predict(matrix_simetrica)

y_train=TRAIN_LABELS
success = 0


# Calculate the time difference
tiempo_ejecucion = time.time()- inicio_tiempo

print(f"Tiempo de ejecución: {tiempo_ejecucion} segundos")
```

Tiempo de ejecución: 0.0006067752838134766 segundos

[52]:
```
y_train=TRAIN_LABELS
success = 0
for i in range(len(y_train)):
    if sol[i] == y_train[i]:
        success += 1

print("Train Accuracy: ", success/len(sol)*100, "%")
```

Precisión del train: 91.90751445086705 %

[53]:
```
print("Checking with test...")
x_test=TEST_DATA
# Record the start time
inicio_tiempo = time.time()
sol = clf.predict(kernel_gram_matrix_full(x_test, x_train))

y_test=TEST_LABELS
success = 0


# Calculate the time difference
```

```
tiempo_ejecucion = time.time()- inicio_tiempo

print(f"Tiempo de ejecución: {tiempo_ejecucion} segundos")
```

```
Comprobando con test…
Calculando matriz de Gram
0 %
0 %
0 %
0 %
0 %
0 %
0 %
1 %
1 %
1 %
1 %
1 %
1 %
1 %
1 %
2 %
2 %
2 %
2 %
2 %
2 %
2 %
2 %
3 %
3 %
3 %
3 %
3 %
3 %
3 %
3 %
4 %
4 %
4 %
4 %
4 %
4 %
4 %
4 %
5 %
5 %
5 %
5 %
5 %
5 %
```

5 %
6 %
6 %
6 %
6 %
6 %
6 %
6 %
7 %
7 %
7 %
7 %
7 %
7 %
7 %
8 %
8 %
8 %
8 %
8 %
8 %
8 %
9 %
9 %
9 %
9 %
9 %
9 %
9 %
10 %
10 %
10 %
10 %
10 %
10 %
10 %
11 %
11 %
11 %
11 %
11 %
11 %
11 %
12 %
12 %
12 %
12 %
12 %

12 %
12 %
13 %
13 %
13 %
13 %
13 %
13 %
13 %
13 %
14 %
14 %
14 %
14 %
14 %
14 %
14 %
14 %
14 %
15 %
15 %
15 %
15 %
15 %
15 %
15 %
15 %
15 %
16 %
16 %
16 %
16 %
16 %
16 %
16 %
16 %
16 %
17 %
17 %
17 %
17 %
17 %
17 %
17 %
17 %
17 %
18 %
18 %
18 %
18 %
18 %
18 %
18 %
18 %
18 %
19 %
19 %
19 %
19 %

19 %
19 %
20 %
20 %
20 %
20 %
20 %
20 %
20 %
20 %
21 %
21 %
21 %
21 %
21 %
21 %
21 %
21 %
22 %
22 %
22 %
22 %
22 %
22 %
22 %
22 %
23 %
23 %
23 %
23 %
23 %
23 %
23 %
23 %
24 %
24 %
24 %
24 %
24 %
24 %
24 %
24 %
25 %
25 %
25 %
25 %
25 %
25 %
25 %
25 %
26 %
26 %
26 %
26 %

26 %
26 %
26 %
27 %
27 %
27 %
27 %
27 %
27 %
27 %
27 %
28 %
28 %
28 %
28 %
28 %
28 %
28 %
28 %
29 %
29 %
29 %
29 %
29 %
29 %
29 %
29 %
30 %
30 %
30 %
30 %
30 %
30 %
30 %
30 %
31 %
31 %
31 %
31 %
31 %
31 %
31 %
31 %
32 %
32 %
32 %
32 %
32 %
32 %
32 %
32 %
33 %
33 %
33 %

33 %
33 %
33 %
33 %
34 %
34 %
34 %
34 %
34 %
34 %
34 %
34 %
35 %
35 %
35 %
35 %
35 %
35 %
35 %
35 %
36 %
36 %
36 %
36 %
36 %
36 %
36 %
36 %
37 %
37 %
37 %
37 %
37 %
37 %
37 %
37 %
38 %
38 %
38 %
38 %
38 %
38 %
38 %
38 %
39 %
39 %
39 %
39 %
39 %
39 %
40 %
40 %
40 %

40 %
40 %
40 %
40 %
41 %
41 %
41 %
41 %
41 %
41 %
41 %
41 %
42 %
42 %
42 %
42 %
42 %
42 %
42 %
42 %
43 %
43 %
43 %
43 %
43 %
43 %
43 %
43 %
44 %
44 %
44 %
44 %
44 %
44 %
44 %
44 %
45 %
45 %
45 %
45 %
45 %
45 %
45 %
45 %
46 %
46 %
46 %
46 %
46 %
46 %
46 %
46 %
47 %
47 %

47 %
47 %
47 %
47 %
47 %
48 %
48 %
48 %
48 %
48 %
48 %
48 %
48 %
49 %
49 %
49 %
49 %
49 %
49 %
49 %
49 %
50 %
50 %
50 %
50 %
50 %
50 %
50 %
50 %
51 %
51 %
51 %
51 %
51 %
51 %
51 %
51 %
52 %
52 %
52 %
52 %
52 %
52 %
52 %
52 %
53 %
53 %
53 %
53 %
53 %
53 %
53 %
53 %
54 %

54 %
54 %
54 %
54 %
54 %
54 %
55 %
55 %
55 %
55 %
55 %
55 %
55 %
55 %
56 %
56 %
56 %
56 %
56 %
56 %
56 %
56 %
57 %
57 %
57 %
57 %
57 %
57 %
57 %
57 %
58 %
58 %
58 %
58 %
58 %
58 %
58 %
58 %
59 %
59 %
59 %
59 %
59 %
59 %
59 %
60 %
60 %
60 %
60 %
60 %
60 %
60 %
61 %

61 %
61 %
61 %
61 %
61 %
61 %
62 %
62 %
62 %
62 %
62 %
62 %
62 %
62 %
63 %
63 %
63 %
63 %
63 %
63 %
63 %
63 %
64 %
64 %
64 %
64 %
64 %
64 %
64 %
64 %
65 %
65 %
65 %
65 %
65 %
65 %
65 %
65 %
66 %
66 %
66 %
66 %
66 %
66 %
66 %
66 %
67 %
67 %
67 %
67 %
67 %
67 %
67 %

68 %
68 %
68 %
68 %
68 %
68 %
68 %
68 %
69 %
69 %
69 %
69 %
69 %
69 %
69 %
69 %
70 %
70 %
70 %
70 %
70 %
70 %
70 %
70 %
71 %
71 %
71 %
71 %
71 %
71 %
71 %
71 %
72 %
72 %
72 %
72 %
72 %
72 %
72 %
72 %
73 %
73 %
73 %
73 %
73 %
73 %
73 %
73 %
74 %
74 %
74 %
74 %
74 %
74 %

74 %
75 %
75 %
75 %
75 %
75 %
75 %
75 %
75 %
76 %
76 %
76 %
76 %
76 %
76 %
76 %
76 %
76 %
77 %
77 %
77 %
77 %
77 %
77 %
77 %
77 %
78 %
78 %
78 %
78 %
78 %
78 %
78 %
78 %
79 %
79 %
79 %
79 %
79 %
79 %
80 %
80 %
80 %
80 %
80 %
80 %
80 %
80 %
81 %
81 %
81 %
81 %
81 %
81 %

```
81 %
82 %
82 %
82 %
82 %
82 %
82 %
82 %
82 %
83 %
83 %
83 %
83 %
83 %
83 %
83 %
83 %
84 %
84 %
84 %
84 %
84 %
84 %
84 %
84 %
85 %
85 %
85 %
85 %
85 %
85 %
85 %
85 %
86 %
86 %
86 %
86 %
86 %
86 %
86 %
86 %
87 %
87 %
87 %
87 %
87 %
87 %
87 %
87 %
88 %
88 %
88 %
88 %
88 %
```

88 %
88 %
89 %
89 %
89 %
89 %
89 %
89 %
89 %
89 %
90 %
90 %
90 %
90 %
90 %
90 %
90 %
90 %
91 %
91 %
91 %
91 %
91 %
91 %
91 %
91 %
92 %
92 %
92 %
92 %
92 %
92 %
92 %
92 %
93 %
93 %
93 %
93 %
93 %
93 %
93 %
93 %
94 %
94 %
94 %
94 %
94 %
94 %
94 %
94 %
95 %
95 %
95 %
95 %

```
95 %
95 %
95 %
96 %
96 %
96 %
96 %
96 %
96 %
96 %
96 %
97 %
97 %
97 %
97 %
97 %
97 %
97 %
97 %
98 %
98 %
98 %
98 %
98 %
98 %
98 %
98 %
99 %
99 %
99 %
99 %
99 %
99 %
Tiempo de ejecución: 1751.9561560153961 segundos
```

[54]:
```python
y_test=TEST_LABELS
success = 0
for i in range(len(y_test)):
    if sol[i] == y_test[i]:
        success += 1

print("Precisión del train: ", success/len(sol)*100, "%")

# Registra el tiempo de finalización
fin_tiempo = time.time()

# Calcula la diferencia de tiempo
tiempo_ejecucion = fin_tiempo - inicio_tiempo

print(f"Tiempo de ejecución de compro test: {tiempo_ejecucion} segundos")
```

```
Precisión del train:  84.17266187050359 %
Tiempo de ejecución de compro test: 1770.4123022556305 segundos
```

[57]:
```
zz_feature_map_reference = ZZFeatureMap(feature_dimension=8, reps=1)# 8 qubits␣
 ↪or n_qubits



def scalar_product_N_q(circuit, x, y):


    zz_feature_map_reference1 = zz_feature_map_reference.assign_parameters(y)

    zz_feature_map_reference_inv = zz_feature_map_reference.assign_parameters(x)

    zzinv1=zz_feature_map_reference_inv.inverse()
    #  Combine the circuits
    circuito_final = zz_feature_map_reference1.compose(zzinv1, qubits=[0,␣
 ↪1,2,3,4,5,6,7])  # for 8 qubits



    backend = Aer.get_backend('statevector_simulator')
    t_qc = transpile(circuito_final, backend)
    qobj = assemble(t_qc)
    result = backend.run(qobj).result()
    statevector = result.get_statevector(t_qc)
    probs = np.abs(statevector)**2
    return probs
```

[58]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle

import warnings
warnings.filterwarnings("ignore")
# constants
n = 2
RANDOM_STATE = 42
LR = 1e-3
class_labels = ['0', '1']
```

```
def normalizeData(DATA_PATH = "./FEATURE_RESULTS/FEATURE_resultante_DP_RAYOMAS.
 ↪csv"):
    """
    Normalizes the data
    """
    # Reads the data
    data = pd.read_csv(DATA_PATH)
    data = shuffle(data, random_state=RANDOM_STATE)
   # For eight qubits the attributes are:
    X, Y = data[['area_pixels', ' mean_coords_x',' mean_coords_y','␣
 ↪centroid_x',' centroid_y',' mean_intensity',' std_intensity',' threshold']].
 ↪values, data[' class'].values
    # normalize the data
    scaler = MinMaxScaler(feature_range=(-0 * np.pi, 2 * np.pi))
    X = scaler.fit_transform(X)
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.8,␣
 ↪random_state=RANDOM_STATE)
    return X_train, X_test, Y_train, Y_test




import qiskit_algorithms
from qiskit_algorithms.optimizers import SPSA

from qiskit import QuantumCircuit

from qiskit.circuit.library import ZZFeatureMap, RealAmplitudes
from qiskit.quantum_info import Statevector

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

from sklearn.utils import shuffle

import warnings
warnings.filterwarnings("ignore")

TRAIN_DATA, TEST_DATA, TRAIN_LABELS, TEST_LABELS = normalizeData()
# Replace all occurrences of 2 with 1
TRAIN_LABELS = np.where(TRAIN_LABELS == 2, 1, TRAIN_LABELS)
TEST_LABELS = np.where(TEST_LABELS == 2, 1, TEST_LABELS)
```

```
#print(TRAIN_DATA)
```

[59]:
```
# para 8 qubits

x_train=TRAIN_DATA[:]
import time
inicio_tiempo = time.time()
matrix_simetrica = kernel_gram_matrix_full_symmetric(x_train, x_train)
fin=time.time()
tiempo=inicio_tiempo-fin
print(tiempo)
```

```
Calculando matriz de Gram
0 %
0 %
1 %
1 %
2 %
2 %
3 %
4 %
4 %
5 %
5 %
6 %
6 %
7 %
8 %
8 %
9 %
9 %
10 %
10 %
11 %
12 %
12 %
13 %
13 %
14 %
15 %
15 %
16 %
16 %
17 %
17 %
18 %
19 %
```

19 %
20 %
20 %
21 %
21 %
22 %
23 %
23 %
24 %
24 %
25 %
26 %
26 %
27 %
27 %
28 %
28 %
29 %
30 %
30 %
31 %
31 %
32 %
32 %
33 %
34 %
34 %
35 %
35 %
36 %
36 %
37 %
38 %
38 %
39 %
39 %
40 %
41 %
41 %
42 %
42 %
43 %
43 %
44 %
45 %
45 %
46 %
46 %

47 %
47 %
48 %
49 %
49 %
50 %
50 %
51 %
52 %
52 %
53 %
53 %
54 %
54 %
55 %
56 %
56 %
57 %
57 %
58 %
58 %
59 %
60 %
60 %
61 %
61 %
62 %
63 %
63 %
64 %
64 %
65 %
65 %
66 %
67 %
67 %
68 %
68 %
69 %
69 %
70 %
71 %
71 %
72 %
72 %
73 %
73 %
74 %

```
75 %
75 %
76 %
76 %
77 %
78 %
78 %
79 %
79 %
80 %
80 %
81 %
82 %
82 %
83 %
83 %
84 %
84 %
85 %
86 %
86 %
87 %
87 %
88 %
89 %
89 %
90 %
90 %
91 %
91 %
92 %
93 %
93 %
94 %
94 %
95 %
95 %
96 %
97 %
97 %
98 %
98 %
99 %
-536.4807784557343
```

[60]:
```
import numpy as np
```

```
# Guardar la matriz en un archivo .npy
np.save('./FEATURE_RESULTS/QKM_RESULTS_EIGHT_QUBITS/
 ↪simulation_eight_q_TRAIN_kernel_matrix_DP_rayomas.npy', matrix_simetrica)
```

[73]:
```
# Ajustar el modelo a los datos de entrenamiento
"""
 encuentra el hiperplano óptimo que separa las diferentes clases en función de␣
 ↪los datos de entrenamiento y las etiquetas asociadas.
 La función objetivo del SVM es convexa, lo que significa que no hay mínimos␣
 ↪locales.
"""
start_time = time.time()
clf.fit(matrix_simetrica, TRAIN_LABELS)
training_time = time.time() - start_time
print("Tiempo de entrenamiento:", training_time, "segundos")
```

```
Tiempo de entrenamiento: 0.0027267932891845703 segundos
```

[62]:
```
#Record the start time
inicio_tiempo = time.time()
sol = clf.predict(matrix_simetrica)

y_train=TRAIN_LABELS
success = 0


# Calculate the time difference
tiempo_ejecucion = time.time()- inicio_tiempo

print(f"Tiempo de ejecución: {tiempo_ejecucion} segundos")
```

```
Tiempo de ejecución: 0.0008602142333984375 segundos
```

[63]:
```
y_train=TRAIN_LABELS
success = 0
for i in range(len(y_train)):
    if sol[i] == y_train[i]:
        success += 1

print("Precisión del train: ", success/len(sol)*100, "%")
```

```
Precisión del train:  100.0 %
```

[77]:
```
zz_feature_map_reference = ZZFeatureMap(feature_dimension=8, reps=1)# 8 qubits␣
 ↪or n_qubits
```

```
def scalar_product_N_q(circuit, x, y):


    zz_feature_map_reference1 = zz_feature_map_reference.assign_parameters(y)

    zz_feature_map_reference_inv = zz_feature_map_reference.assign_parameters(x)

    zzinv1=zz_feature_map_reference_inv.inverse()
    # Combinar los circuitos
    circuito_final = zz_feature_map_reference1.compose(zzinv1, qubits=[0,␣
 ↪1,2,3,4,5,6,7])  # for 3 qubits



    backend = Aer.get_backend('statevector_simulator')
    t_qc = transpile(circuito_final, backend)
    qobj = assemble(t_qc)
    result = backend.run(qobj).result()
    statevector = result.get_statevector(t_qc)
    probs = np.abs(statevector)**2
    return probs


def kernel_gram_matrix_full(X1, X2):
    print("Calculando matriz de Gram")

    gram_matrix = np.zeros((X1.shape[0], X2.shape[0]))
    for i, x1 in enumerate(X1):
        print(int(i / len(X1) * 100), "%")
        for j, x2 in enumerate(X2):
            qc = QuantumCircuit(8, 8)
            x1 = x1.flatten()
            x2 = x2.flatten()
            prob = scalar_product_N_q(qc,x1, x2)[0]  # Usar solo la␣
 ↪probabilidad del estado '00'
            gram_matrix[i, j] = prob

    return gram_matrix
```

[78]:
```
print("Comprobando con test...")
x_test=TEST_DATA
# Registra el tiempo de inicio
inicio_tiempo = time.time()
sol = clf.predict(kernel_gram_matrix_full(x_test, x_train))

y_test=TEST_LABELS
success = 0
```

```python

# Calcula la diferencia de tiempo
tiempo_ejecucion = time.time()- inicio_tiempo

print(f"Tiempo de ejecución: {tiempo_ejecucion} segundos")
```

```
Comprobando con test…
Calculando matriz de Gram
0 %
0 %
0 %
0 %
0 %
0 %
0 %
1 %
1 %
1 %
1 %
1 %
1 %
1 %
2 %
2 %
2 %
2 %
2 %
2 %
2 %
2 %
3 %
3 %
3 %
3 %
3 %
3 %
3 %
3 %
4 %
4 %
4 %
4 %
4 %
4 %
4 %
5 %
5 %
5 %
```

5 %
5 %
5 %
5 %
6 %
6 %
6 %
6 %
6 %
6 %
6 %
7 %
7 %
7 %
7 %
7 %
7 %
7 %
8 %
8 %
8 %
8 %
8 %
8 %
8 %
9 %
9 %
9 %
9 %
9 %
9 %
9 %
10 %
10 %
10 %
10 %
10 %
10 %
10 %
11 %
11 %
11 %
11 %
11 %
11 %
11 %
12 %
12 %

```
12 %
12 %
12 %
12 %
12 %
13 %
13 %
13 %
13 %
13 %
13 %
13 %
14 %
14 %
14 %
14 %
14 %
14 %
14 %
14 %
15 %
15 %
15 %
15 %
15 %
15 %
15 %
16 %
16 %
16 %
16 %
16 %
16 %
16 %
16 %
17 %
17 %
17 %
17 %
17 %
17 %
17 %
17 %
18 %
18 %
18 %
18 %
18 %
18 %
18 %
19 %
```

19 %
19 %
19 %
19 %
19 %
20 %
20 %
20 %
20 %
20 %
20 %
20 %
20 %
21 %
21 %
21 %
21 %
21 %
21 %
21 %
21 %
22 %
22 %
22 %
22 %
22 %
22 %
22 %
22 %
23 %
23 %
23 %
23 %
23 %
23 %
23 %
23 %
24 %
24 %
24 %
24 %
24 %
24 %
24 %
24 %
25 %
25 %
25 %
25 %
25 %
25 %
25 %
25 %
26 %

26 %
26 %
26 %
26 %
26 %
26 %
27 %
27 %
27 %
27 %
27 %
27 %
27 %
28 %
28 %
28 %
28 %
28 %
28 %
28 %
28 %
29 %
29 %
29 %
29 %
29 %
29 %
29 %
29 %
30 %
30 %
30 %
30 %
30 %
30 %
30 %
31 %
31 %
31 %
31 %
31 %
31 %
31 %
31 %
32 %
32 %
32 %
32 %
32 %
32 %
32 %

33 %
33 %
33 %
33 %
33 %
33 %
33 %
33 %
34 %
34 %
34 %
34 %
34 %
34 %
34 %
34 %
34 %
35 %
35 %
35 %
35 %
35 %
35 %
35 %
35 %
35 %
36 %
36 %
36 %
36 %
36 %
36 %
36 %
36 %
36 %
37 %
37 %
37 %
37 %
37 %
37 %
37 %
37 %
37 %
38 %
38 %
38 %
38 %
38 %
38 %
38 %
38 %
38 %
39 %
39 %
39 %
39 %
39 %
39 %

40 %
40 %
40 %
40 %
40 %
40 %
40 %
41 %
41 %
41 %
41 %
41 %
41 %
41 %
41 %
42 %
42 %
42 %
42 %
42 %
42 %
42 %
42 %
43 %
43 %
43 %
43 %
43 %
43 %
43 %
43 %
44 %
44 %
44 %
44 %
44 %
44 %
44 %
44 %
45 %
45 %
45 %
45 %
45 %
45 %
45 %
45 %
46 %
46 %
46 %
46 %
46 %
46 %

46 %
47 %
47 %
47 %
47 %
47 %
47 %
47 %
47 %
48 %
48 %
48 %
48 %
48 %
48 %
48 %
48 %
48 %
49 %
49 %
49 %
49 %
49 %
49 %
49 %
49 %
50 %
50 %
50 %
50 %
50 %
50 %
50 %
50 %
51 %
51 %
51 %
51 %
51 %
51 %
51 %
51 %
52 %
52 %
52 %
52 %
52 %
52 %
52 %
52 %
53 %
53 %
53 %
53 %
53 %

53 %
53 %
54 %
54 %
54 %
54 %
54 %
54 %
54 %
54 %
55 %
55 %
55 %
55 %
55 %
55 %
55 %
55 %
56 %
56 %
56 %
56 %
56 %
56 %
56 %
56 %
57 %
57 %
57 %
57 %
57 %
57 %
57 %
57 %
58 %
58 %
58 %
58 %
58 %
58 %
58 %
58 %
59 %
59 %
59 %
59 %
59 %
59 %
59 %
60 %
60 %
60 %
60 %
60 %

60 %
60 %
61 %
61 %
61 %
61 %
61 %
61 %
61 %
61 %
62 %
62 %
62 %
62 %
62 %
62 %
62 %
62 %
62 %
63 %
63 %
63 %
63 %
63 %
63 %
63 %
63 %
64 %
64 %
64 %
64 %
64 %
64 %
64 %
64 %
65 %
65 %
65 %
65 %
65 %
65 %
65 %
65 %
66 %
66 %
66 %
66 %
66 %
66 %
66 %
66 %
67 %
67 %
67 %
67 %

67 %
67 %
67 %
68 %
68 %
68 %
68 %
68 %
68 %
68 %
68 %
69 %
69 %
69 %
69 %
69 %
69 %
69 %
69 %
70 %
70 %
70 %
70 %
70 %
70 %
70 %
70 %
71 %
71 %
71 %
71 %
71 %
71 %
71 %
71 %
72 %
72 %
72 %
72 %
72 %
72 %
72 %
72 %
72 %
73 %
73 %
73 %
73 %
73 %
73 %
73 %
73 %
73 %
74 %
74 %
74 %

74 %
74 %
74 %
74 %
75 %
75 %
75 %
75 %
75 %
75 %
75 %
75 %
76 %
76 %
76 %
76 %
76 %
76 %
76 %
76 %
77 %
77 %
77 %
77 %
77 %
77 %
77 %
77 %
78 %
78 %
78 %
78 %
78 %
78 %
78 %
78 %
79 %
79 %
79 %
79 %
79 %
79 %
80 %
80 %
80 %
80 %
80 %
80 %
80 %
80 %
81 %
81 %
81 %

81 %
81 %
81 %
81 %
82 %
82 %
82 %
82 %
82 %
82 %
82 %
82 %
83 %
83 %
83 %
83 %
83 %
83 %
83 %
83 %
84 %
84 %
84 %
84 %
84 %
84 %
84 %
84 %
85 %
85 %
85 %
85 %
85 %
85 %
85 %
85 %
86 %
86 %
86 %
86 %
86 %
86 %
86 %
86 %
87 %
87 %
87 %
87 %
87 %
87 %
87 %
87 %
88 %
88 %

88 %
88 %
88 %
88 %
88 %
89 %
89 %
89 %
89 %
89 %
89 %
89 %
89 %
90 %
90 %
90 %
90 %
90 %
90 %
90 %
90 %
91 %
91 %
91 %
91 %
91 %
91 %
91 %
91 %
92 %
92 %
92 %
92 %
92 %
92 %
92 %
92 %
93 %
93 %
93 %
93 %
93 %
93 %
93 %
93 %
94 %
94 %
94 %
94 %
94 %
94 %
94 %
94 %
95 %

```
95 %
95 %
95 %
95 %
95 %
95 %
96 %
96 %
96 %
96 %
96 %
96 %
96 %
96 %
97 %
97 %
97 %
97 %
97 %
97 %
97 %
97 %
98 %
98 %
98 %
98 %
98 %
98 %
98 %
98 %
98 %
99 %
99 %
99 %
99 %
99 %
99 %
Tiempo de ejecución: 4395.329967260361 segundos
```

[79]:
```python
y_test=TEST_LABELS
success = 0
for i in range(len(y_test)):
    if sol[i] == y_test[i]:
        success += 1

print("Precisión del test: ", success/len(sol)*100, "%")
```

```
Precisión del test:  99.42446043165467 %
```

[ ]:
```python

```