Library Import

In [1]:

```
# Importing standard Qiskit libraries
from qiskit import QuantumCircuit, transpile,BasicAer
from qiskit.tools.jupyter import *
from qiskit.visualization import *
# from ibm_quantum_widgets import *
from qiskit_aer import AerSimulator
from qiskit.algorithms.optimizers import SPSA
import pandas as pd
from qiskit.circuit import ParameterVector
from qiskit.visualization import circuit_drawer


# qiskit-ibmq-provider has been deprecated.
# Please see the Migration Guides in https://ibm.biz/provider_migration_gu
from qiskit_ibm_runtime import QiskitRuntimeService, Sampler, Estimator, S

# Loading your IBM Quantum account(s)
# service = QiskitRuntimeService(channel="ibm_quantum")
from pylab import cm
from sklearn import metrics
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import numpy as np
from qiskit.circuit.library import ZZFeatureMap
from qiskit_machine_learning.kernels import TrainableFidelityQuantumKernel
from qiskit_machine_learning.kernels.algorithms import QuantumKernelTraine
from qiskit_machine_learning.algorithms import QSVC
from qiskit_machine_learning.kernels import QuantumKernel
# Invoke a primitive inside a session. For more details see https://qiskit
# with Session(backend=service.backend("ibmq_qasm_simulator")):
#     result = Sampler().run(circuits).result()
```

In [2]:

```
# Importing standard Qiskit libraries
from qiskit import QuantumCircuit, transpile, BasicAer, IBMQ, execute, Aer
from qiskit.tools.jupyter import *
from qiskit.visualization import *
from qiskit import Aer
from qiskit.algorithms.optimizers import SPSA
from qiskit.circuit import ParameterVector
from qiskit.visualization import circuit_drawer

# General libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pylab import cm
from sklearn import metrics

# QKT related
from qiskit.circuit.library import ZZFeatureMap
from qiskit_machine_learning.algorithms import QSVC
from qiskit_machine_learning.kernels import FidelityQuantumKernel, Trainab
# from qiskit_machine_learning.kernels.algorithms import QuantumKernelTrai
from qiskit.algorithms.state_fidelities import ComputeUncompute
# from qiskit.primitives import Sampler

# Additional imports
from sklearn.decomposition import PCA
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split

from collections import Counter
```

In [3]:

```
class QKTCallback:
    """Callback wrapper class."""

    def __init__(self) -> None:
        self._data = [[] for i in range(5)]

    def callback(self, x0, x1=None, x2=None, x3=None, x4=None):
        """
        Args:
            x0: number of function evaluations
            x1: the parameters
            x2: the function value
            x3: the stepsize
            x4: whether the step was accepted
        """
        self._data[0].append(x0)
        self._data[1].append(x1)
        self._data[2].append(x2)
        self._data[3].append(x3)
        self._data[4].append(x4)

    def get_callback_data(self):
        return self._data

    def clear_callback_data(self):
        self._data = [[] for i in range(5)]
```

Preparing the Dataset

In [4]:
```
df = pd.read_csv(r'200 EXAMPLES.csv')
ytrain= df.iloc[:,7]
Xtrain= df.iloc[:,1:7]
Xtrain
```

Out[4]:

	Coolant temp	Fuel pressure	LOG(Engine rpm)	LOG(Lub oil pressure)	LOG(Coolant pressure)	lub oil temp
0	-0.772790	0.344223	0.482768	-0.243775	0.243036	0.563408
1	-0.086326	-0.062279	-0.316501	-0.244287	0.517765	0.034793
2	-0.385653	-0.147072	0.213726	-0.628313	-0.312446	0.898516
3	0.490692	0.434899	-0.460128	-0.058842	-0.961579	0.445002
4	0.580827	0.158654	-0.119384	-0.074489	-0.187463	-0.559568
...	...	...	...	...	...	...
195	0.172962	-0.287470	0.507028	-0.450261	-0.991773	0.520044
196	0.538044	0.534762	0.783982	-0.667502	-0.003361	0.209038
197	0.139001	-0.308990	0.109317	-0.973335	-0.717331	0.987307
198	-0.692401	-0.287310	0.089582	-0.116380	-0.163690	0.428278
199	-0.048245	-0.380093	0.635449	0.370611	0.764489	0.336724

200 rows × 6 columns

In [ ]:
```

```

In [5]:
```
#To reduce training dataset size
Xtrain,Xtest , ytrain, ytest = train_test_split(Xtrain, ytrain,stratify=yt
```

In [6]:
```
Xtrain
```

Out[6]:

	Coolant temp	Fuel pressure	LOG(Engine rpm)	LOG(Lub oil pressure)	LOG(Coolant pressure)	lub oil temp
83	0.629259	-0.174393	-0.149220	-0.600644	0.018092	-0.007425
20	0.903901	-0.247201	-0.865533	-0.697440	0.232856	-0.654695
190	0.873835	-0.883777	0.472312	-0.853440	-0.761545	0.106029
13	0.207256	0.427178	-0.343775	0.467497	0.579474	-0.430303
90	-0.487668	-0.605603	0.486246	0.630593	-0.144128	-0.225894
...	...	...	...	...	...	...
52	0.640812	0.122278	0.336653	-0.699092	0.335254	-0.169100
176	-0.267394	-0.448664	0.255256	0.073995	-0.196576	0.098360
93	0.402057	-0.097079	0.085620	-0.415191	-0.163029	0.363089
151	-0.403449	0.062842	0.651950	0.108193	-0.066501	0.078752
1	-0.086326	-0.062279	-0.316501	-0.244287	0.517765	0.034793

150 rows × 6 columns

In [7]:
```
ytrain
```

Out[7]:
```
83     1
20     0
190    0
13     0
90     1
      ..
52     0
176    1
93     0
151    0
1      1
Name: Engine Condition, Length: 150, dtype: int64
```

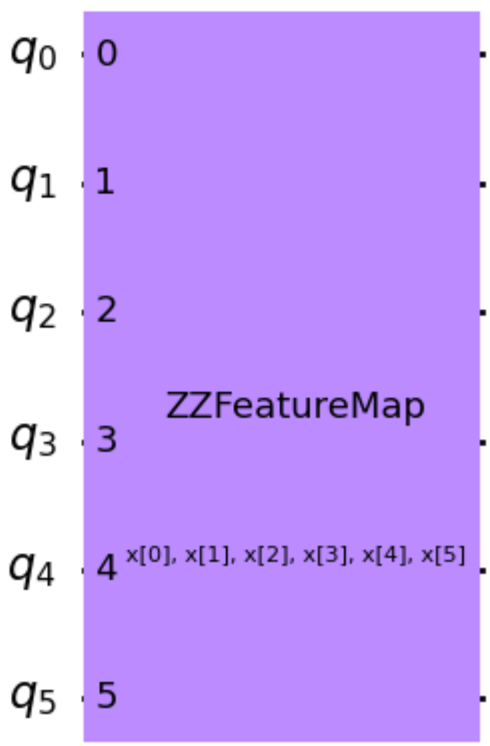
Define the Quantum Feature Map

In [8]:

```
from qiskit.circuit.library import PauliFeatureMap,ZFeatureMap
from qiskit.visualization import circuit_drawer

# Define the parameter vector
training_params = ParameterVector("θ", 1)

# Define the quantum circuit with PauliFeatureMap
fm0 = QuantumCircuit(6)
for i in range(6):
    fm0.rx(training_params[0], i)

# Define the PauliFeatureMap


# Create the feature map, composed of the two circuits
# fm = fm0.compose(fm1)
fm=ZZFeatureMap(6, entanglement='linear')
# Print the circuit and trainable parameters
print(circuit_drawer(fm, output='mpl'))
print(f"Trainable parameters: {training_params}")
fm.draw(output="mpl")
```

```
Figure(371.107x535.111)
Trainable parameters: θ, ['θ[0]']
```

Out[8]:



In [9]:
```
qc=ZZFeatureMap(6, entanglement='circular')
print(transpile(qc,optimization_level=3).depth())
```

```
40
```

In [11]:
```
# Connecting with IBMQ
from qiskit_ibm_runtime import QiskitRuntimeService, Sampler, Estimator, S
# Loading your IBM Quantum account(s)
service = QiskitRuntimeService(channel="ibm_quantum")
```

In [12]:
```

service.backends()
```

Out[12]:
```
[<IBMBackend('ibmq_manila')>,
 <IBMBackend('ibmq_qasm_simulator')>,
 <IBMBackend('ibmq_quito')>,
 <IBMBackend('simulator_mps')>,
 <IBMBackend('simulator_statevector')>,
 <IBMBackend('simulator_stabilizer')>,
 <IBMBackend('ibm_lagos')>,
 <IBMBackend('ibmq_lima')>,
 <IBMBackend('ibmq_belem')>,
 <IBMBackend('simulator_extended_stabilizer')>,
 <IBMBackend('ibm_nairobi')>,
 <IBMBackend('ibm_perth')>,
 <IBMBackend('ibmq_jakarta')>]
```

In [13]:
```
# Use the IBM quantum backend
backend = service.backend("ibmq_qasm_simulator")
```

In [ ]:

In [ ]:

In [ ]:

In [14]:
```
from qiskit_aer.noise import NoiseModel

noisy_backend = service.get_backend('ibmq_jakarta')
backend_noise_model = NoiseModel.from_backend(noisy_backend)

simulator = service.get_backend('ibmq_qasm_simulator')

options = Options()
options.resilience_level = 0
options.optimization_level = 0
options.simulator = {
    "noise_model": backend_noise_model
}
with Session(service=service, backend=simulator) as session:
    sampler = Sampler(session=session, options=options)
    fidelity = ComputeUncompute(sampler=sampler)
    quantum_kernel = FidelityQuantumKernel(fidelity=fidelity, feature_map=
    qsvc = QSVC(quantum_kernel=quantum_kernel)
    qsvc.fit(Xtrain, ytrain)
```

In [ ]:

In [ ]:

In [ ]:

Fit and Test the Model

In [15]:
```
# # Use QSVC for classification
# qsvc = QSVC(quantum_kernel=optimized_kernel)

# # Fit the QSVC
# qsvc.fit(Xtrain, ytrain)
```

In [16]:
```

# Predict the labels
labels_test = qsvc.predict(Xtest)

# Evalaute the test accuracy
accuracy_test = metrics.balanced_accuracy_score(y_true=ytest, y_pred=label
print(f"accuracy test: {accuracy_test}")
```

```
accuracy test: 0.46
```

In [17]:
```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(ytest,labels_test))
print(classification_report(ytest,labels_test))
```

```
[[13 12]
 [15 10]]
              precision    recall  f1-score   support

           0       0.46      0.52      0.49        25
           1       0.45      0.40      0.43        25

    accuracy                           0.46        50
   macro avg       0.46      0.46      0.46        50
weighted avg       0.46      0.46      0.46        50
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Visualize the Kernel Training Process

In [ ]:

In [ ]: