One of the key challenges in classification with SVM is to find the optimal hyperplane. Quantum Computing is facilitating the optimization.

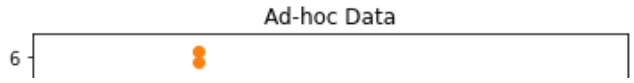
A key concept in classification methods is that of a kernel. Data cannot typically be separated by a hyperplane in its original space. A common technique used to find such a hyperplane consists of applying a non-linear transformation function to the data. This function is called a feature map, as it transforms the raw features, or measurable properties, of the phenomenon or subject under study. Classifying in this new feature space -and, as a matter of fact, also in any other space, including the raw original one- is nothing more than seeing how close data points are to each other. This is the same as computing the inner product for each pair of data points in the set. So, in fact we do not need to compute the non-linear feature map for each datum, but only the inner product of each pair of data points in the new feature space. This collection of inner products is called the kernel and it is perfectly possible to have feature maps that are hard to compute but whose kernels are not.

Source: Vojtech Havlicek, Antonio D. C´orcoles, Kristan Temme, Aram W. Harrow, Abhinav Kandala, Jerry M. Chow, and Jay M. Gambetta1, “Supervised learning with quantum enhanced feature spaces,” arXiv: 1804.11326

```
!pip install qiskit
!pip install qiskit.aqua.components.feature_maps
```

```
import matplotlib.pyplot as plt
import numpy as np

from qiskit import BasicAer
from qiskit.circuit.library import ZZFeatureMap
from qiskit.aqua import QuantumInstance, aqua_globals
from qiskit.aqua.algorithms import QSVM
from qiskit.aqua.utils import split_dataset_to_data_and_labels, map_label_to_class_name

seed = 10599
aqua_globals.random_seed = seed
from qiskit.ml.datasets import ad_hoc_data, sample_ad_hoc_data

feature_dim = 2
sample_total, training_input, test_input, class_labels = ad_hoc_data(
    training_size=20,
    test_size=10,
    n=feature_dim,
    gap=0.3,
    plot_data=True
)
extra_test_data = sample_ad_hoc_data(sample_total, 10, n=feature_dim)
datapoints, class_to_label = split_dataset_to_data_and_labels(extra_test_data)
print(class_to_label)
```

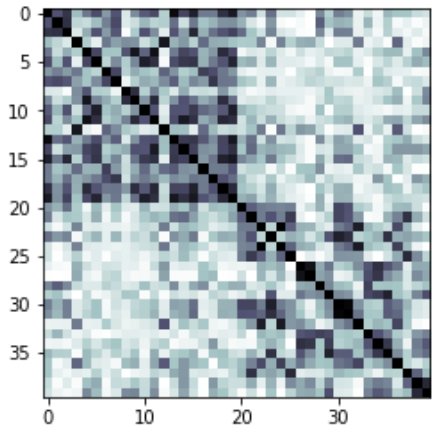
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:11: DeprecationWarning: The variable qiskit.aqua.aqua_globals is c
  # This is added back by InteractiveShellApp.init_path()
/usr/local/lib/python3.7/dist-packages/qiskit/ml/__init__.py:40: DeprecationWarning: The package qiskit.ml is deprecated. It was
  warn_package('ml', 'qiskit_machine_learning', 'qiskit-machine-learning')
```




```
1  feature_map = ZZFeatureMap(feature_dimension=feature_dim, reps=2, entanglement='linear')
2  qsvm = QSVM(feature_map, training_input, test_input, datapoints[0])
3
4  backend = BasicAer.get_backend('qasm_simulator')
5  quantum_instance = QuantumInstance(backend, shots=1024, seed_simulator=seed, seed_transpiler=seed)
6
7  result = qsvm.run(quantum_instance)
8
9  print(f'Testing success ratio: {result["testing_accuracy"]}')
10 print()
11 print('Prediction from datapoints set:')
12 print(f'  ground truth: {map_label_to_class_name(datapoints[1], qsvm.label_to_class)}')
13 print(f'  prediction:   {result["predicted_classes"]}')
14 predicted_labels = result["predicted_labels"]
15 print(f'  success rate: {100*np.count_nonzero(predicted_labels == datapoints[1])/len(predicted_labels)}%')
```

```
/usr/local/lib/python3.7/dist-packages/qiskit/aqua/algorithms/classifiers/qsvm/qsvm.py:104: DeprecationWarning: The package qiskit.aqua
  'qiskit-machine-learning')
/usr/local/lib/python3.7/dist-packages/qiskit/aqua/quantum_instance.py:137: DeprecationWarning: The class qiskit.aqua.QuantumInstance
  'qiskit-terra')
Testing success ratio: 1.0

Prediction from datapoints set:
  ground truth: ['A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B']
  prediction:   ['A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B']
  success rate: 100.0%
```

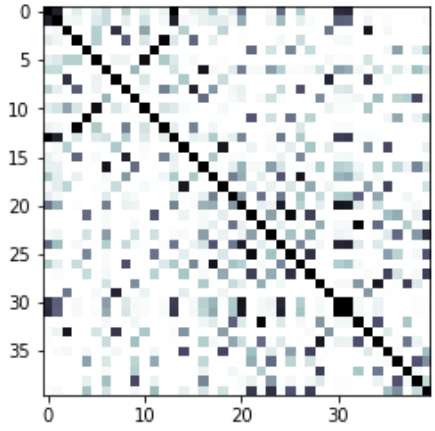
```
1  kernel_matrix = result['kernel_matrix_training']
2  plt.imshow(np.asmatrix(kernel_matrix),interpolation='nearest',origin='upper',cmap='bone_r');
```



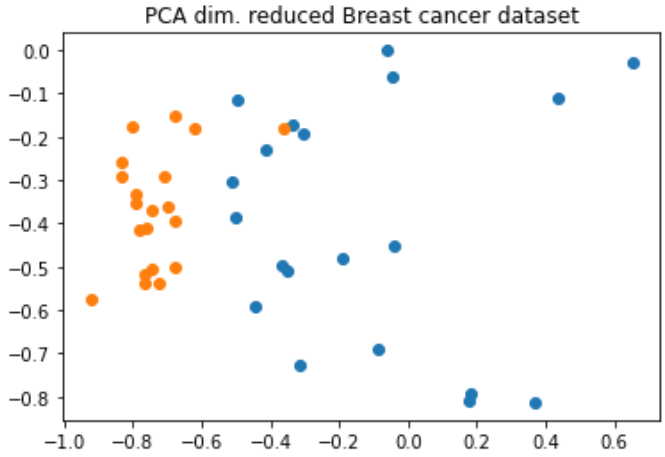

```
1  from qiskit.aqua.algorithms import SklearnSVM
2
3  result = SklearnSVM(training_input, test_input, datapoints[0]).run()
4
5  print(f'Testing success ratio: {result["testing_accuracy"]}')
6  print()
7  print('Prediction from datapoints set:')
8  print(f'  ground truth: {map_label_to_class_name(datapoints[1], qsvm.label_to_class)}')
9  print(f'  prediction:   {result["predicted_classes"]}')
10 predicted_labels = result["predicted_labels"]
11 print(f'  success rate: {100*np.count_nonzero(predicted_labels == datapoints[1])/len(predicted_labels)}%')
```

```
kernel_matrix = result['kernel_matrix_training']
plt.imshow(np.asmatrix(kernel_matrix), interpolation='nearest', origin='upper', cmap='bone_r');
```

```
Testing success ratio: 0.65

Prediction from datapoints set:
  ground truth: ['A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B', 'B']
  prediction:   ['B', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'A', 'A', 'A', 'B', 'B', 'B', 'A', 'B', 'A', 'B', 'B', 'A']
  success rate: 55.0%
```




```
from qiskit.ml.datasets import breast_cancer

feature_dim = 2
sample_total, training_input, test_input, class_labels = breast_cancer(
    training_size=20,
    test_size=10,
    n=feature_dim,
    plot_data=True)
```

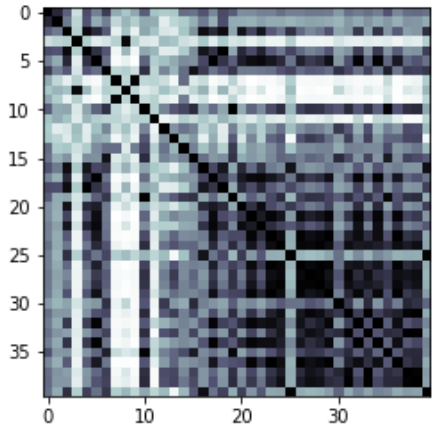



```
feature_map = ZZFeatureMap(feature_dimension=feature_dim, reps=2, entanglement='linear')
qsvm = QSVM(feature_map, training_input, test_input)

backend = BasicAer.get_backend('qasm_simulator')
quantum_instance = QuantumInstance(backend, shots=1024, seed_simulator=seed, seed_transpiler=seed)

result = qsvm.run(quantum_instance)

print(f'Testing success ratio: {result["testing_accuracy"]}')
```

```
Testing success ratio: 0.9
```

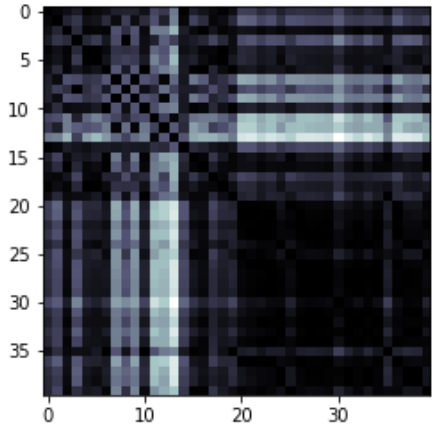
```
kernel_matrix = result['kernel_matrix_training']
```

```
2  img = plt.imshow(np.asmatrix(kernel_matrix),interpolation='nearest',origin='upper',cmap='bone_r')
```




```
1  result = SklearnSVM(training_input, test_input).run()
2
3  print(f'Testing success ratio: {result["testing_accuracy"]}')
4
5  kernel_matrix = result['kernel_matrix_training']
6  plt.imshow(np.asmatrix(kernel_matrix), interpolation='nearest', origin='upper', cmap='bone_r');
```

```
Testing success ratio: 0.85
```



✓ 0s completed at 7:24 AM