# Quantum Support Vector Machines (qSVMs) for Cancerous Cell Detection

Alice Liu

# Table of Contents

# Abstract

Around 3.8 million women in the US are diagnosed with breast cancer. In terms of the survival rate, late or misclassified diagnosis decreases an individual's chance of survival from 90% to just 15%. With the rise of quantum computing that relies on mechanical properties such as superposition, entanglement and tanging, this will allow for speedups in optimization methods. Quantum machine learning, a hybrid of quantum computing and AI, enables faster and more accurate processing of data compared to our current classical machine learning techniques, which has the potential for detecting breast cancer at an earlier stage. This may lead to better treatment options and a higher chance of survival. This specific proposal utilizes the quantum support vector machines (qSVM), a supervised learning algorithm, to classify breast cancer cells as benign or malignant based on set numerical parameters of the cells. Within the qSVM algorithm, there will be a 2-qubit simulation, 4-qubit simulation and 8-qubit simulation run on IBM's hardware. The classification accuracy for these simulations are compared to the accuracy from a classical support vector machine algorithm acting as the control. The method that employs the quantum support vector machine algorithm with a 2-qubit simulation yielded the highest result, with a classification accuracy of 90%.

# I. Introduction

Developments made in the emerging field of quantum computing and specifically quantum machine learning have been exponentially increasing over the past few years [1]. In 2019 Google declared reaching "quantum supremacy" with a superconducting processor that was able to create quantum states on 53 qubits, while IonQ introduced the first commercial trapped-ion quantum computers with a program length of 60 two-qubit gates. Both of these developments introduced increased speedups and decreased decoherence in information processing on quantum hardware. Despite these developments, much more work needs to be done before quantum computers can be deployed commercially and to a wide scale audience. High cost, bulkiness, decoherence of qubits, and restricted environments and temperatures for operation are a few limitations present within today's quantum computers. Quantum computers at this state can be compared to computers and the Internet in the 1970s, in terms of scalability potential and cost. Similar to our current computers today, quantum computers also follow the trend of Moore's Law, which states that the number of transistors per silicon chip doubles each year.

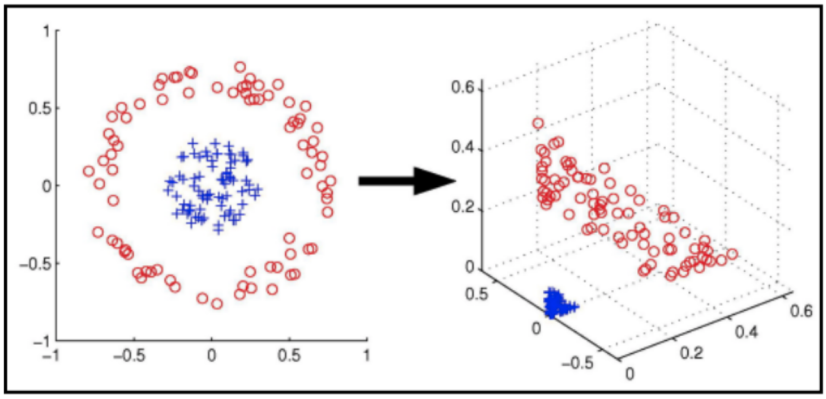
Quantum computing itself is characterized by the use of quantum phenomena, such as superposition, entanglement and tunneling to perform computations. Quantum machine learning is a subset within quantum computing that attempts to bridge the gap

between quantum computing and machine learning, a subset of Artificial Intelligence (AI). This specific area exploits the benefits of the quantum mechanical properties or "quantum advantages," as well as the established algorithmic models and computational infrastructure in current machine learning developments.

Due to superposition and entanglement, the main advantage for quantum computers comes from how they can process a vast number of calculations simultaneously. While our current computers that we use leverage bit strings of 1s and 0s, which are evaluated one by one, quantum computers leverage qubit strings of 1s, 0s, and superpositions of 1s and 0s (10, 01). This allows for $2^n$ simulations to be completed all at the same time. For example, a 100 qubit computer can run $2^{100}$ calculations simultaneously. This makes quantum computing ideal for applications including optimization, as well as machine learning methods. In this experiment, the quantum support vector machine algorithm (SVM) is utilized to explore the applied benefits of quantum machine learning.

The SVM is a supervised learning algorithm, where given labeled training data, will output an optimal hyperplane able to categorize new examples. In other words, within a two-dimensional space for example, it finds a line dividing the plane into two parts, where one class sits on each side.

Finding the optimal hyperplane however becomes much more difficult as more dimensions are added, when applied to 3D, 4D and even 100D.




As shown above, adding a third axis z makes the identification of the hyperplane more difficult. However, by doing various transformations and adding a third axis z, we are able to plot these points in the z-axis, and we are now easily able to draw a plane, as shown in the above image on the right. The kernel trick was used to do this, which is the mapping of a non-linear data set into a higher dimensional space where the hyperplane is found to separate the samples.

But what if the dimensions of the data points are projected to keep getting higher and higher as the dataset becomes increasingly complex? In this case, it would be very difficult for classical computers to operate through these large computations. Even if the classical computer was capable of computing through this, time is another issue.

Depending on the number of examples used, features and regularization parameters, it can take up to weeks to train.

This is where quantum machine learning comes in, harnessing the power of quantum computing to accelerate the training process with accurate classification even with complex datasets in high dimensions. There are quantum interpretations of the SVM kernel trick, allowing the reduction of calculations for a particular dimension and allowing the splitting of these high-dimensional datasets into more manageable ones.

Quantum machine learning in this case is essentially the analysis of classical data and quantum states on a quantum computer. The whole essence of it is being able to compute immense quantities of data more intelligently and quickly, providing the computational advantage of classifying objects too complex for classical computers and more thorough data analysis. The quantum SVM algorithm takes the classical machine learning algorithm and performs the support vector machine on a quantum circuit in order to be efficiently processed on a quantum computer. With kernel methods, there are currently limitations when the feature space becomes larger. Kernel methods become computationally expensive to estimate. Quantum algorithms offer speed-ups with the properties of entanglement and interference, allowing for the exploitation of an exponentially large quantum feature and state space.

This method specifically represents the feature state of a classification problem by a quantum state by taking advantage of the quantum Hilbert space (which has a large dimension). A quantum kernel estimator is used, which estimates the kernel function and optimizes the classifier directly. A quantum processor is used to estimate the kernel function of the quantum feature space directly, then implement the conventional SVM.

The process of the algorithm is as follows:

1) Data is provided classically and the quantum state space is used as a feature space. Data is mapped to a quantum state by applying the feature map circuit to a reference state.
2) A short-depth quantum circuit is applied to the feature state. The circuit with $l$ layers is parameterized and will be optimized during training.
3) For a two label classification {-1, +1} a binary measurement is applied to the quantum state.
4) For the decision rule: perform R repeated measurement shots to obtain the empirical distribution. For optimization, a cost function needs to be defined. The empirical risk is defined given by the error probability of assigning the incorrect label averaged over the samples in the training set $T$.

For this specific experiment, the quantum SVM method for the classification in the dataset of breast cancer cells as benign or malignant is leveraged. Within the quantum

SVM algorithm, there will be a 2-qubit simulation, 4-qubit simulation and 8-qubit simulation. Each of the three simulations are the independent variables. All three scenarios will be compared to a (conventional) SVM method used in classical machine learning, which acts as the control. The classification accuracy (expressed as a percentage) for these cells is the dependent variable.

The hypothesis for this research is that the quantum SVM algorithm would have a higher classification accuracy than the classical SVM algorithm because of the "quantum advantages" including 1) faster runtime 2) greater capacity and 3) higher efficiency that the quantum algorithm holds. In addition, an increase in the number of qubits would increase the accuracy due to how a greater number of qubits allows for more information to be efficiently processed all at once.

# II. Materials & Methods

In this experiment, the accuracy in the binary classification of breast cancer cells would likely be higher under the use of a quantum-enabled support vector machine algorithm run on a quantum circuit and processor, as opposed to a support vector machine algorithm run classically. An increased number of qubits, from 2, 4, 8 qubits will also likely increase the accuracy of the classification in these cells.

This experiment was conducted solely on Python with the assistance of a few other quantum-based software packages. The full list of the materials used are as follows:

- Modern Operating System consisting of x86 64-bit CPU (Intel / AMD architecture). For this experiment, the Mac OS Mojave Version 10.14.5 with a 1.1 GHz Intel Core m3 was used.
- Python Version 3.7.4
- Scikit Learn Software Package Version 0.21.3. This is a machine learning library for Python that holds classification, regression and clustering algorithms.
- Interactive web-based computational environment. Example: Jupyter Notebook.
- IBM Quantum Cloud Service API. This is leveraged to run the algorithms and test data points on IBM's quantum hardware processors using a Python Interface.

- IBM's Qiskit Software Development Kit Version 0.23.4. This provides the tools for creating and manipulating circuits and gates while deploying quantum-based algorithms and application modules.
- IBM's Qiskit Packages:
    - Qiskit Aqua Version 0.8.1. This package provides numerous algorithms for implementations in domains such as machine learning and AI, chemistry, optimization and finance.
    - Qiskit Aer Version 0.7.3. This package provides a framework for the execution of quantum circuits with optimized simulator backends as well as the tools for configurable noise models.
    - Qiskit QCGPU Provider Version 0.2.0. This package contains quantum circuit simulators including the Statevector (returns the state vector of a quantum circuit applied in the |0> state) and Qasm Simulator (simulates a compiled QASM quantum circuit).
- UCI Machine Learning Repository Breast Cancer Wisconsin (Diagnostic) Data

When implementing the experiment, the first step is importing the necessary data packages directly from the Jupyter Notebook host. These imports are directly from the Qiskit Aqua, Aer and Scikit Learn packages to aid in the data importation, preparation/organization and analysis processes.

```
#Visualization & Data Importing Tools
import numpy as np
import pandas as pd
import scipy
import seaborn as sns
from scipy.linalg import expm
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

#Data Preparation & Classical Machine Learning Tools
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.decomposition import PCA
from qiskit.aqua.utils import split_dataset_to_data_and_labels

#Quantum Processing Tools
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
```

*Importing the needed packages and pre-processing tools*

The dataset that will be used for the classification between cancerous and non-cancerous breast cancer cells is imported. This specific dataset contains 31 different parameters or features for each cell nucleus, each having an ID number with the diagnosis type (M = malignant, B = benign). For the experiment purposes, the diagnosis as benign or malignant will be used as the target while 5 out of the 31 features will be used for determining the diagnosis type when training the model.

The features (parameters) used include:

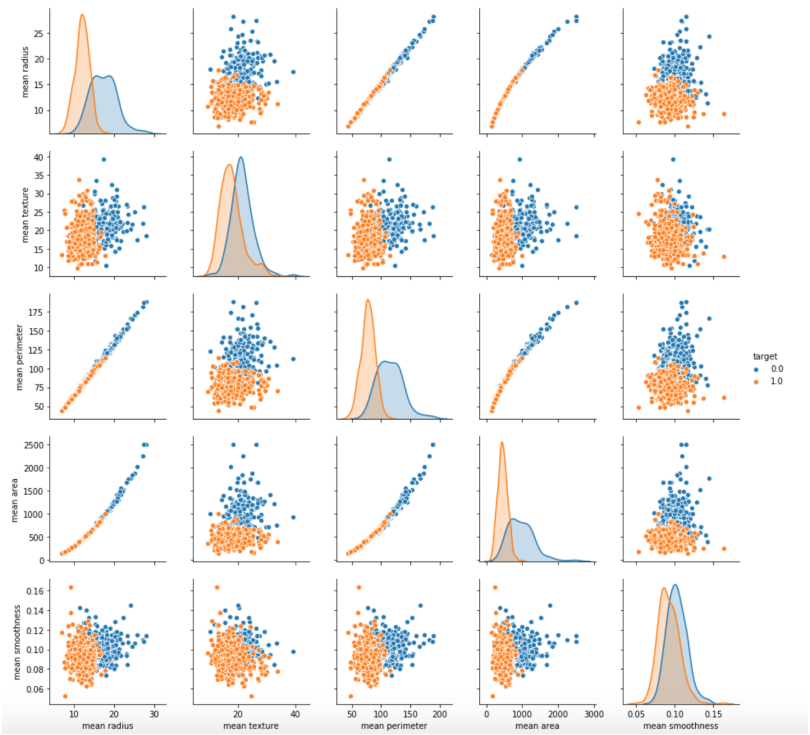
1) Radius (the mean of the distances from the center to the points on the perimeter)
2) Texture (a standard deviation of the gray-scale values)
3) Perimeter
4) Area
5) Smoothness (local variation in the radii lengths)

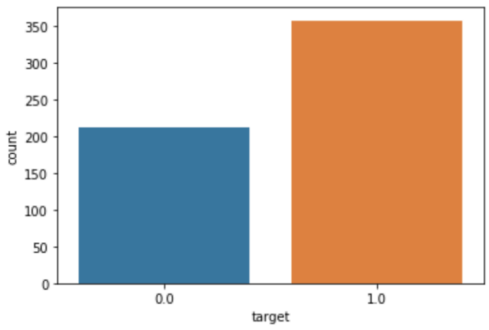
The dataset is imported by loading the root path of the UCI Breast Cancer Dataset. Once the data is loaded, the data is preprocessed and organized so that the features are the parameters and the diagnosis value is the target.

```
#Loading the dataset:
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()

#Preprocessing of the data:
df_cancer = pd.DataFrame(np.c_[cancer['data'], cancer['target']], columns =
                         np.append(cancer['feature_names'], ['target']))
sns.pairplot(df_cancer, hue = 'target', vars = ['mean radius', 'mean texture',
                                                'mean perimeter','mean area','mean smoothness'] )
```

The data will finish its preprocessing and adapt the data into the models it uses once the correlations between the parameters are defined. The relationships between the variables used, which types of data had the most heavy influence on each other, and the actual number of benign and malignant cases in the dataset can be displayed visually:




*Orange = Malignant*

*Blue = Benign*

With the dataset loaded, the data can now be processed in preparation for being inputted into the algorithm. The steps are as follows:

1) Divide the dataset into training and testing portions to test the accuracy of the classifier. The data is divided into 70% training and 30% testing.
2) Standardize the data set's features to fit into a normal distribution.
3) Use Principal Component Analysis (PCA) to reduce and fit the number of dimensions in the dataset into an *n* number of qubits used. For the experiment purposes, 2, 4, and 8 qubits were used. This is necessary for the algorithm to find patterns from the base of the given number of qubits while keeping variation.
4) Scale data between -1 and 1 to set a range for the Support Vector Machine model.
5) Pick a sample to train the model from.

```
def breast_cancer(training_size, test_size, n, PLOT_DATA=True):
    class_labels = [r'Benign', r'Malignant']

    #testing classifier accuracy
    #training: 70%, testing: 30%
    X_train, X_test, Y_train, Y_test = train_test_split(cancer.data, cancer.target,
                                                        test_size=0.3, random_state=109)

    #Standardize dataset features to fit a normal distribution
    scaler = StandardScaler().fit(X_train)
    X_train = scaler.transform(X_train)
    X_test = scaler.transform(X_test)

    #Use PCA to break down data from 30 to 'n' dimensions
        #(finds patterns while keeping variation)
    pca = PCA(n_components=n).fit(X_train)
    X_train = pca.transform(X_train)
    X_test = pca.transform(X_test)

    # Scaling the data to be between -1 and 1
    samples = np.append(X_train, X_test, axis=0)
    minmax_scale = MinMaxScaler((-1, 1)).fit(samples)
    X_train = minmax_scale.transform(X_train)
    X_test = minmax_scale.transform(X_test)

    # Picking sample to train model from:
    training_input = {key: (X_train[Y_train == k, :])[:training_size] for k,
                      key in enumerate(class_labels)}
    test_input = {key: (X_train[Y_train == k, :])[training_size:(
        training_size+test_size)] for k, key in enumerate(class_labels)}
```

*Method of implementation for the data training and testing with standardization and preprocessing*

```
if PLOT_DATA:
    for k in range(0, 2):
        x_axis_data = X_train[Y_train == k, 0][:training_size]
        y_axis_data = X_train[Y_train == k, 1][:training_size]

        label = 'Malignant' if k is 1 else 'Benign'
        plt.scatter(x_axis_data, y_axis_data, label=label)

    plt.title("Breast Cancer Dataset (With PCA)")
    plt.legend()
    plt.show()


return X_train, training_input, test_input, class_labels
```

*Method of implementation for the PCA before adapting the model into the SVM algorithm.*

With the data preprocessed and trained, the classical (conventional) support vector machine algorithm, which acts as the control variable, can be implemented to classify the cells into benign and malignant groups. A confusion matrix and kernel matrix are also outputted to better visualize the success ratios and its distributions.

```
# Loading the model
from sklearn.svm import SVC
svc_model = SVC()

# Fitting the model into the SVM algorithm
svc_model.fit(X_train_scaled, y_train)

# Predictions
y_predict = svc_model.predict(X_test_scaled)

# Importing the metric libraries
from sklearn.metrics import classification_report, confusion_matrix

# Confusion matrix:
cm = np.array(confusion_matrix(y_test, y_predict, labels=[1,0]))
confusion = pd.DataFrame(cm, index=['is_malignant', 'is_benign'],
                         columns=['predicted_malignant','predicted_benign'])
print(confusion)
sns.heatmap(confusion,annot=True,fmt="d")


```

*Method for classical SVM implementation.*

*The confusion matrix is also implemented.*

```
# CLASSICAL VERSION
# Comparison using SVM RBF Kernel
result = SVM_Classical(training_input, test_input, datapoints[0]).run()
print("kernel matrix during the training:")
kernel_matrix = result['kernel_matrix_training']
img = plt.imshow(np.asmatrix(kernel_matrix),interpolation='nearest',
                 origin='upper',cmap='bone_r')
plt.show()

print("testing success ratio: ", result['testing_accuracy'])

print("ground truth: {}".format(map_label_to_class_name(datapoints[1], label_to_class)))
print("predicted:    {}".format(result['predicted_classes']))
```

*Comparison of the success ratios with the classical SVM algorithm using a kernel matrix.*

The quantum version of the SVM algorithm can now be implemented. The quantum support vector machine takes the classical data from the earlier preprocessing and uses a quantum kernel estimator. This estimates the kernel function and optimizes the classifier directly. With the assistance of IBM Quantum's API, it can be leveraged as a quantum processor for the algorithm to run, which estimates the kernel function of the quantum feature space directly, then implements the conventional SVM.

The steps to implementing the quantum SVM algorithm are:

1) Set the dimensionality and number of qubits the circuit will have
2) Initialize the feature map in order to build the quantum SVM
3) Set the necessary parameters for algorithm training, including the depth of the circuit, number of shots and initializing the pseudo-random number generator

```
from qiskit import IBMQ
IBMQ.load_account()
token = 'api_token_#'

feature_dim = 4
sample_total, training_input, test_input, class_labels = breast_cancer(
    training_size=20,
    test_size=10,
    n=feature_dim,
    plot_data=True
)

temp = [test_input[k] for k in test_input]
total_array = np.concatenate(temp)

aqua_dict = {
    'problem': {'name': 'classification', 'random_seed': 100},
    'algorithm': {
        'name': 'QSVM'
    },
    'backend': {'provider': 'qiskit.BasicAer', 'name': 'qasm_simulator', 'shots': 256},
    'feature_map': {'name': 'SecondOrderExpansion', 'depth': 2, 'entanglement': 'linear'}
}

algo_input = ClassificationInput(training_input, test_input, total_array)
result = run_algorithm(aqua_dict, algo_input)

for k,v in result.items():
    print("'{}' : {}".format(k, v))
```

*In order to access a quantum processing unit, the token needs to be initialized from the user's IBM Quantum cloud account.*

*feature_dim refers to the number of qubits.*

*This will change from 2,4,8 qubits. The quantum SVM algorithm is imported and the parameters are defined, including the provider type, number of shots and depth*

The algorithm is now able to run, the run method completing the training, testing and prediction of the unlabeled data. The ground truth, final predictions, prediction class, success ratio and accuracy are then generated. To better visualize the outcome, a kernel matrix can be built directly from the training sample of the dataset after the algorithm was implemented.

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

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

*Kernel matrix implementation for the quantum SVM algorithm*

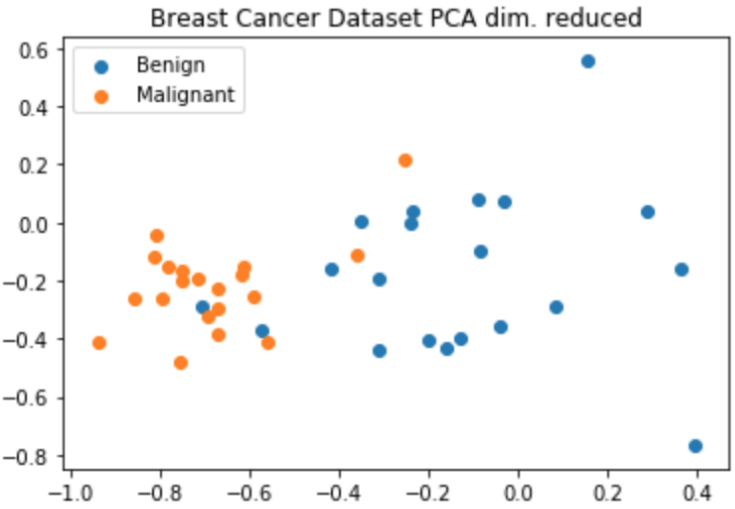
# III. Results

After the running of the model and algorithm, the output of the results can be shown. With the classical support vector machine algorithm (control variable), a classification accuracy of 85% was outputted. With the running of the quantum support vector machine algorithm, three different instances were implemented including a 2-qubit, 4-qubit and 8-qubit (independent variables) simulation with the use of IBM Quantum's cloud processing unit. In the 2-qubit simulation, the binary classification accuracy between benign and malignant cells was 90%. The 4-qubit simulation yielded a classification accuracy of 75% while the 8-qubit simulation yielded a classification accuracy of 65%.

To summarize, the classification accuracies are outputted in the table below. The classification accuracy refers to the binary classification between benign and malignant groups based on the 5 features used in training.

| Model Type | Classification Accuracy |
|---|---|
| Classical Support Vector Machine (SVM) Algorithm | 85% |
| Quantum Support Vector Machine (qSVM) Algorithm, 2-Qubits | 90% |

| | |
|---|---|
| Quantum Support Vector Machine (qSVM) Algorithm, 4-Qubits | 75% |
| Quantum Support Vector Machine (qSVM) Algorithm, 8-Qubits | 65% |



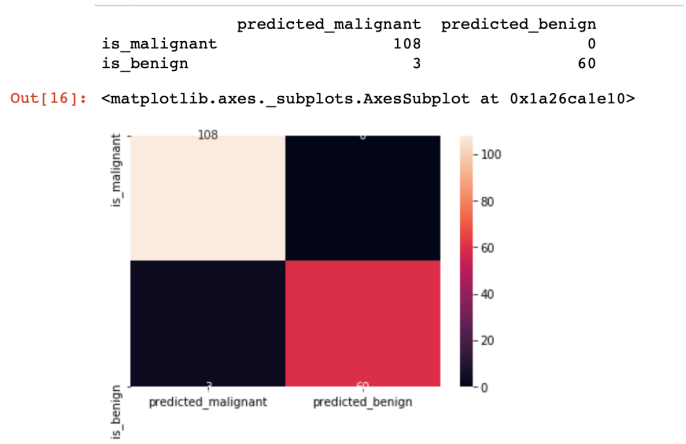

```
{'Benign': 0, 'Malignant': 1} {0: 'Benign', 1: 'Malignant'}
```

*PCA Plot*

The Principal Component Analysis (PCA) Plot from the preprocessing of the data is displayed. PCA is a dimensionality-reduction method used to reduce the dimensionality of a large data set. The method is able to transform a large set of variables into a smaller one, increasing interpretability while at the same time minimizing information loss.

In this use case with the breast cancer dataset, the data was reduced from 5 parameters to 2 principal components. The benign and malignant groups as shown have 2 separate

clusters from each other, indicating that the features that each group possesses are distinct from the other group. For example, the radius, texture, perimeter, area and smoothness throughout the group of cancerous cells classified as benign are all of similar value. The same can also be said for the cells classified as malignant. On the other hand, when comparing benign to malignant or malignant to benign, the features between the group are of differing value.




*Correlation Matrix*

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0.0 | 1.00 | 0.95 | 0.98 | 63 |
| 1.0 | 0.97 | 1.00 | 0.99 | 108 |
| accuracy | | | 0.98 | 171 |
| macro avg | 0.99 | 0.98 | 0.98 | 171 |
| weighted avg | 0.98 | 0.98 | 0.98 | 171 |

*Performance Measures*

The correlation matrix as well as its performance measures including precision, recall, F1 score and support are displayed from the classical SVM algorithm. In the correlation matrix, the total number of incorrect predictions is 3, where there were 3 instances when the SVM incorrectly predicted the cell was malignant (and was actually benign).

In the performance measures, there are 4 different combinations: True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN).

- TP: correctly predicted positive values. The value of the actual class is yes and the value of the predicted class is also yes.
- TN: correctly predicted negative values. Actual class: no. Predicted class: no.
- FP: falsely predicted positive values. Actual class: no. Predicted class: yes.
- FN: falsely predicted negative values. Actual class: yes. Predicted class: no.

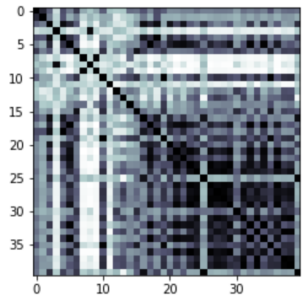
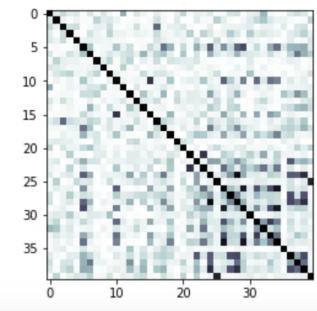
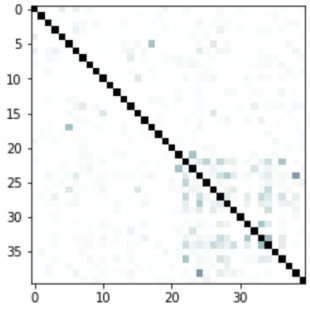
The precision is the ratio of the correctly predicted positive observations to the total predicted positive observations. Precision mathematically is equivalent to TP/TP+FP. High precision relates to low false positive rate. In this case, the precision rate is very high for the malignant and benign cells, at 100% and 97% respectively. Recall is the ratio of correctly predicted positive observations to all the observations in the class. Recall mathematically is equivalent to TP/TP+FN. The recall for both the malignant and benign groups are also high, at 95% and 100% respectively. F1 score is the weighted average of Precision and Recall as another method to measure accuracy. This score takes both false positives and false negatives into account. The F1 score mathematically is equivalent to 2 * (Recall * Position) / (Recall + Precision). The F1 score for the malignant group is 98% while the benign group is 100%.


*Correlation Matrix for the Classical SVM Algorithm*

Despite the high individual parameter scores when training with the precision, recall and F1, the testing success ratio when testing the algorithm was much lower. Overall, the algorithm outputted a classification accuracy of 85% for the testing success ratio.

```
'predicted_labels' : [0 1 0 0 0 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1]
'predicted_classes' : ['Benign', 'Malignant', 'Benign', 'Benign', 'Benign', 'Benign', 'Malign
ant', 'Malignant', 'Malignant', 'Benign', 'Malignant', 'Malignant', 'Benign', 'Malignant', 'M
alignant', 'Benign', 'Malignant', 'Malignant', 'Malignant', 'Malignant']
```

When running the quantum support vector machine algorithm, the individual vectors during the testing process and SVM kernel matrix training are outputted. After the training and testing process finishes with the quantum algorithm, the predicted labels and classes are outputted. The kernel matrix training and testing process for the displayed case was during the 2-qubit simulation.

| 2 Qubits | 4 Qubits | 8 Qubits |
|---|---|---|
| Testing success ratio: 0.9  | Testing success ratio: 0.75  | Testing success ratio: 0.65  |

The success ratios are then outputted. The 2-qubit simulation resulted in a testing success ratio of 90%, the 4-qubit simulation resulted in a testing success ratio of 75%,

while the 2-qubit simulation resulted in a testing success ratio of 65%. It appears that as the number of qubits used in testing the quantum SVM algorithm increased, the testing success ratio gradually decreased.

Overall, in terms of classification accuracy with identifying cancerous cells in benign and malignant groups, the quantum SVM algorithm with 2 qubits yielded the highest number of cancerous cells correctly classified, while the classical SVM algorithm yielded the lowest number of cancerous cells correctly classified.

# IV. Discussion

In this work, an example for employing quantum machine learning was shown in order to demonstrate the advantages of quantum methods in terms of their speedups and depth in classification. These are known as "quantum advantages," where quantum mechanical properties such as tunneling, superposition and entanglement allow for such speedups compared to today's computers.

By combining the intersection of state-of-the-art, high-potential quantum computing and conventional AI and machine learning, this allows for a greater speedup in terms of runtime and a greater classification accuracy. The interplay of machine learning algorithms with the analysis of classical data executed on a quantum processor allows for the infrastructure support on the quantum computing part (AI infrastructure for data processing and analysis is more established). This also allows for "advantages" enabled by the laws of quantum mechanics to also be applied to the machine learning part.

The original hypothesis for this research was that the quantum SVM algorithm would have a higher classification accuracy than the classical SVM algorithm. In addition, an increase in the number of qubits would increase the accuracy due to how a greater number of qubits allows for more information to be efficiently processed all at once. For example, the 2-qubit simulation will allow for 4 calculations to be completed at once, the

4-qubit simulation will allow for 16 calculations and the 8-qubit simulation will allow for 256 calculations.

The hypothesis was partially supported. The quantum SVM algorithm for the 2-qubit simulation did indeed have a higher classification than the classical SVM algorithm. The quantum SVM algorithm had a classification accuracy of 90% while the classical SVM algorithm had a classification accuracy of 85%. However, as the number of qubits increased, the classification accuracy decreased. Both the 4-qubit and 8-qubit simulations yielded a lower classification accuracy than the classical SVM algorithm, with accuracies of 75% and 65%, respectively.

This is likely due to the current hardware limitations that quantum computers have. In this case, the calculations were run through IBM's quantum processors over the cloud. Though an increased number of qubits allows for more efficient processing due to an increased number of simulations able to be run, this will also become more susceptible to noise. An increased amount of noise within the running algorithm will decrease the accuracy. However, the processing speed with the quantum SVM algorithm is significantly faster than that of the classical SVM algorithm. The quantum SVM has an algorithmic complexity logarithmic in feature size and number of training data. This can be represented with O(logmn) where m is the sample size and n is the dimension of each data point. The classical algorithm on the other hand requires polynomial time for both

training and prediction. Learning efficiency and the capacity for the quantum SVM algorithm are also greater than the classical SVM.

# V. Conclusion

In this work, a successful implementation of a quantum machine learning method, the quantum support vector machine algorithm was demonstrated with a 2-qubit, 4-qubit and 8-qubit simulation. In terms of efficiency, the quantum support vector machine algorithm held a higher rate. With quantum computing as a new and emerging field, with quantum machine learning even more recent, the state-of-the-art system has much potential in the next coming decades as hardware developments are being made for quantum error correction and noise reduction. Quantum machine learning has many implications ranging from medicine, drug discovery, genomics, security and beyond. A few areas that have the potential to be boosted by quantum machine learning are: 1) chemical simulation, including mapping out the molecules and atoms for the creation of new materials 2) quantum matter simulation, including modeling molecular interactions at an atomic level, allowing new pharmaceuticals and medical research and 3) quantum communication networks for the transmission of more secure data.

This implementation is growing ever more efficient with the increase in the number of qubits available, and will be able to classify these large and complex datasets at a lower computational cost than what is currently available with classical computers.

# **VI.** References


1. Havlicek, Vojtech, et al. "Supervised Learning with Quantum Enhanced Feature Spaces." *ArXiv.org*, 5 June 2018, arxiv.org/abs/1804.11326.
2. Mohseni, Masoud, et al. *Quantum Support Vector Machine for Big Data Classification*, American Physical Society, 1 Feb. 2014, dspace.mit.edu/handle/1721.1/90391.