Quantum Machine Learning Classification

Fabio Hinojosa Jimenez

Darryl R Chajon

Quantum Information Science

Final Project Report

12/08/2024

Fall 2024

Abstract

Quantum Machine Learning (QML) has emerged as a promising approach to address challenges in error classification by harnessing quantum principles such as superposition and entanglement. These principles offer the potential to enhance computational efficiency, particularly as quantum systems grow in complexity and susceptibility to noise. This study explores the application of QML algorithms to error classification tasks, comparing their performance to classical machine learning techniques. With the increasing demand for scalable and effective quantum error classification methods, our research examines the current capabilities and limitations of QML in this domain.

Through extensive simulations and experiments, we find that QML models currently deliver performance comparable to classical methods, with limitations in both speed and precision. However, our results indicate that using more diverse datasets and incorporating more accurate error simulations—or leveraging actual quantum hardware—could yield significant advantages over their classical counterparts. These insights suggest that while QML has not yet surpassed classical methods in this field, it holds substantial promise for future developments, warranting further exploration and refinement.

Introduction

Quantum computing represents a paradigm shift in computational power and efficiency, harnessing the principles of quantum mechanics to tackle complex problems that are infeasible for classical computers. However, one of the primary challenges facing quantum computing today is quantum error correction, as quantum systems are highly susceptible to noise and decoherence. These errors are induced faults that disrupt quantum states and compromise computations. The development of effective error mitigation and correction techniques is critical to the realization of practical quantum computers. This issue necessitates the development of methods to detect and classify errors in quantum computations to ensure the reliability of quantum algorithms.

In this context, both **classical machine learning (CML)** and **quantum machine learning (QML)** have shown potential in improving error classification methods. Classical machine learning, with its established algorithms and techniques, has been employed to detect, classify, and mitigate errors in quantum systems. On the other hand, quantum machine learning, which leverages quantum computing's unique properties, introduces new avenues for error classification by potentially outperforming classical approaches in terms of computational efficiency and the ability to model complex quantum systems.

This paper explores and compares the strengths and limitations of quantum machine learning with its classical counterparts in the domain of **error classification**. By reviewing current research and methods, we aim to highlight how these two approaches can be integrated or stand alone in addressing the challenges of error classification, and the potential future directions that might shape the evolution of quantum error correction techniques.

Methods

The first and most important tool utilized in this study was qiskit, which is an open-source software development kit used for working with quantum computers at the level of circuits, pulses, and algorithms. It provides tools for creating and manipulating quantum programs and running them on prototype quantum devices on the IBM Quantum Platform or on simulators on a local computer. Qiskit primarily uses the Python programming language. Next, we utilized scikit-learn which is an open-source machine learning library for Python that features various classification, regression and clustering algorithms including support-vector machines that are designed to interoperate with the very important scientific libraries Numpy and SciPy, which are libraries utilized for high performance

linear algebra and array operations. Another important library we utilized is Matplotlib, which is a comprehensive library for creating static, animated, and interactive visualizations of data in Python. Lastly, we utilized AerSimulator which is a high-performance quantum circuit simulator within the Qiskit framework, designed to accurately model quantum computing operations with realistic noise models allowing users to simulate complex quantum circuits on their local systems with high efficiency.

Data

The dataset we used to train our machine learning models was from the UC Irvine Machine Learning Repository called the “wine” dataset, a table of 178 instances of wine with quantitative values of 13 features, and 3 classes of identification. This dataset was selected to prioritize the learning aspects over extensive data preprocessing.

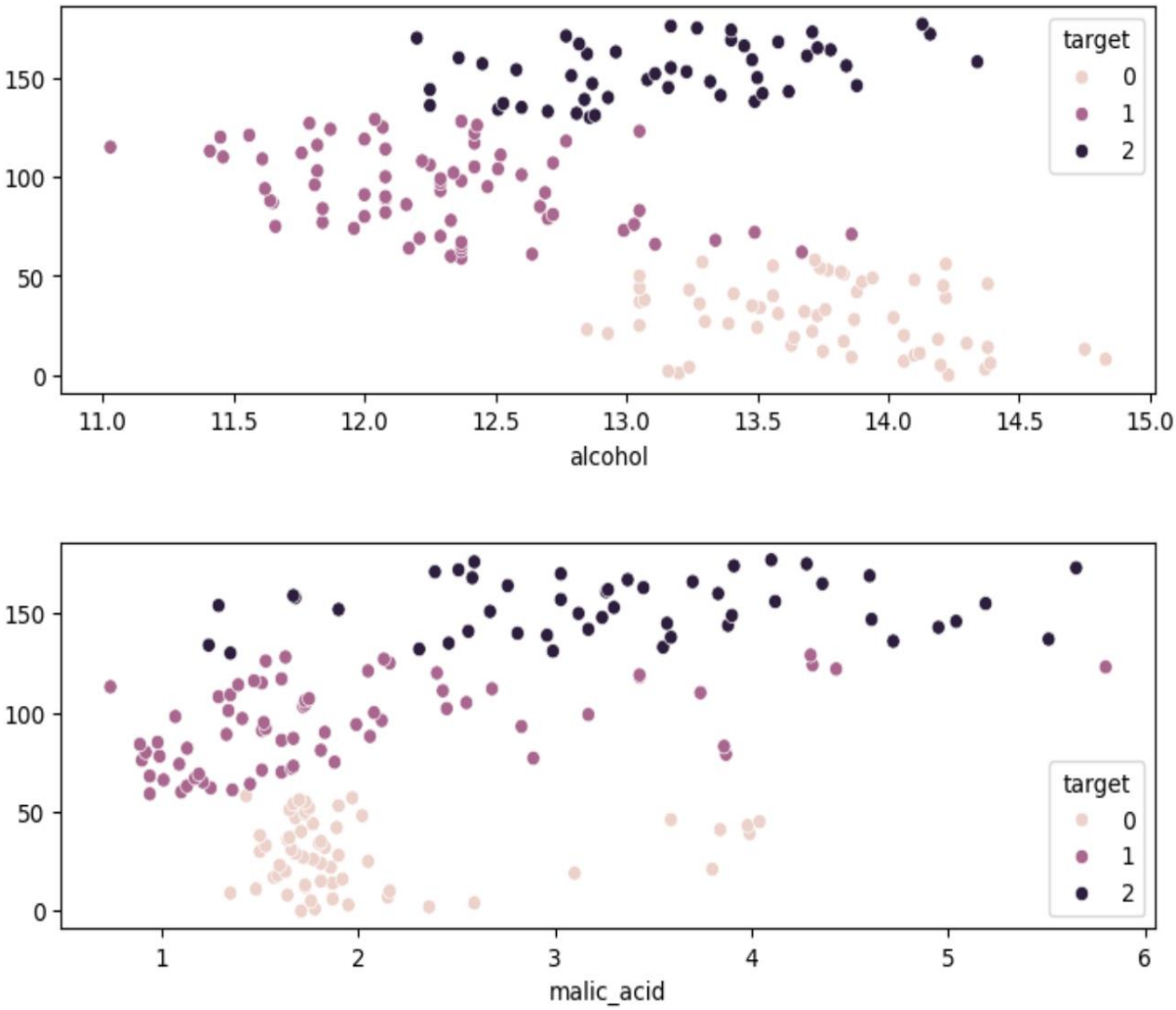

Figure 1. Two plots from wine dataset of 2 different features: malic acid and alcohol content

As can be seen by figure 1 above, it is observable to the human eye that there is linear separability between the 3 classes of wine relative to their individual features. This trend continues for every feature of wine recorded in this dataset.

```
# Separating features for both datasets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
X_train.shape, X_test.shape
```

```
((142, 13), (36, 13))
```

Figure 2. Code example of data separation

For training and testing, the dataset was first split into 80% for training and 20% for testing, then repeated with a split of 70% training and 30% testing. To standardize the data, we applied a Min-Max Scaler, ensuring compatibility with quantum machine learning (QML) algorithms. A quantum kernel was employed to map classical feature vectors into a Hilbert space, enabling the use of quantum computing techniques. More specifically, Qiskit's "ZZFeatureMap" was utilized, and through experimentation, we determined that a fully entangled feature map, in contrast to a linear entangled map, produced the most optimal results for this application.

Models

For the approach of this project, we decided to use a variety of different models to compare them with each other, the following results were obtained by utilizing a Support Vector Machine (SVM), Quantum Support Vector Machine (QSVM), a Simple multiclass Neural Network (NN), and a Variational Quantum Classifier Quantum Neural Network (VQC QNN).

Support Vector Machine (SVM)

The Support Vector Machine is a supervised machine learning algorithm that classifies data by finding an optimal line or hyperplane that maximizes the distance between each class in an N-dimensional space. SVMs are commonly used for problems involving classification and logical regression. The SVM algorithm is widely used in machine learning as it can handle both linear and nonlinear classification tasks. However, when the data is not linearly separable, kernel functions are used to transform data into higher-dimensional space to enable linear separation. This application of kernel functions is known as the "kernel trick", and the choice of kernel function, such as linear kernels, polynomial kernels, radial basis function (RBF) kernels, or sigmoid kernels, depends on data characteristics and the specific use case.

Quantum Support Vector Machine (QSVM)

Quantum Support Vector Machines (QSVMs) are the quantum analog to classical support vector machines. QSVMs are more efficient than traditional models because they can process complex datasets and identify patterns that might be difficult for traditional models. QSVMs utilize quantum algorithms within the SVM framework to analyze classical data or run on quantum devices to supplement classical machine learning programs.

Neural Network (NN)

Neural networks (NN) are computational models inspired by the human brain, used in Artificial Intelligence (AI) to process data by mimicking the interconnected structure of neurons, where information is passed through layers of nodes to learn and make predictions based on patterns within data. Each artificial neuron receives signals from connected neurons, then processes them and sends a signal to other connected neurons. The "signal" is a real number, and the output of each neuron is computed by some non-linear function of the sum of its inputs, called the activation function. The strength of the signal at each connection is determined by a weight, which adjusts during the learning process. Neurons are aggregated into layers; different layers perform different transformations on their inputs. Signals travel from the first layer (the input layer) to the last layer (the output layer), possibly passing through multiple intermediate layers (hidden layers). A network is typically called a deep neural network if it has at least two hidden layers. Artificial neural networks are used for various tasks, including predictive modeling, adaptive control, and solving problems in artificial intelligence.

Quantum Neural Network (QNN)

Quantum neural networks (QNNs) are the quantum analog to neural networks. Most Quantum neural networks are developed as feed-forward networks. Similar to their classical counterparts, this structure intakes input from one layer of qubits, and passes that input onto another layer of qubits. This layer of qubits evaluates this information and passes on the output to the next layer. Eventually the path leads to the final layer of qubits. The layers do not have to be of the same width, meaning they don't have to have the same number of qubits as the layer before or after it. QNNs can be theoretically trained similarly to training classical/artificial neural networks. A key difference lies in communication between the layers of neural networks. For classical neural networks, at the end of a given operation, the current perceptron (an algorithm for supervised learning of classifiers) copies its output to the next layer of perceptrons in the network. However, in a quantum neural network, where each perceptron is a qubit, this would violate the no-cloning theorem. A proposed generalized solution to this is to replace the classical fan-out method

with an arbitrary unitary that spreads out, but does not copy, the output of one qubit to the next layer of qubits. Using this fan-out Unitary (Uf) with an ancilla bit in a known state, the information from the qubit can be transferred to the next layer of qubits. This process adheres to the quantum operation requirement of reversibility.

Results

```
#svm with rbf kernel
svc=SVC(C=100.0)
svc.fit(X_train,y_train)
y_pred=svc.predict(X_test)
print('Accuracy score with rbf kernel and C=100.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Accuracy score with rbf kernel and C=100.0 : 1.0000
```

```
#svm with linear kernel
linear_svc=SVC(kernel='linear', C=1.0)
linear_svc.fit(X_train,y_train)
y_pred_test=linear_svc.predict(X_test)
print('Accuracy score with linear kernel and C=1.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred_test)))
```

```
Accuracy score with linear kernel and C=1.0 : 1.0000
```

```
#svm with polynomial kernel
poly_svc=SVC(kernel='poly', C=1.0)
poly_svc.fit(X_train,y_train)
y_pred=poly_svc.predict(X_test)
print('Accuracy score with polynomial kernel and C=1.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Accuracy score with polynomial kernel and C=1.0 : 0.9444
```

```
#svm with polynomial kernel and C=100
poly_svc100=SVC(kernel='poly', C=100.0)
poly_svc100.fit(X_train,y_train)
y_pred=poly_svc100.predict(X_test)
print('Accuracy score with polynomial kernel and C=100.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Accuracy score with polynomial kernel and C=100.0 : 0.9722
```

```
#svm with sigmoid kernel
sigmoid_svc=SVC(kernel='sigmoid', C=1.0)
sigmoid_svc.fit(X_train,y_train)
y_pred=sigmoid_svc.predict(X_test)
print('Accuracy score with sigmoid kernel and C=1.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Accuracy score with sigmoid kernel and C=1.0 : 0.9722
```

Figure 3. SVM results with different kernels and 80/20 training/testing split

```
QSVC Score: 0.9722222222222222

Classification Report:
              precision    recall  f1-score   support

           1       1.00      0.93      0.96        14
           2       0.94      1.00      0.97        16
           3       1.00      1.00      1.00         6

    accuracy                           0.97        36
   macro avg       0.98      0.98      0.98        36
weighted avg       0.97      0.97      0.97        36
```

Figure 4. QSVM score with 80/20 training/test split

```
X_train: (142, 13)
y_train: (142,)
X_test: (36, 13)
y_test: (36,)
Classification Report:
              precision    recall  f1-score   support

           0       0.93      1.00      0.97        14
           1       1.00      0.88      0.93        16
           2       0.86      1.00      0.92         6

    accuracy                           0.94        36
   macro avg       0.93      0.96      0.94        36
weighted avg       0.95      0.94      0.94        36
```

Figure 5. Multiclass NN with 80/20 training/test split

Figure 6. VQC QNN results

```
#svm with rbf kernel
svc=SVC(C=100.0)
svc.fit(X_train,y_train)
y_pred=svc.predict(X_test)
print('Accuracy score with rbf kernel and C=100.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Accuracy score with rbf kernel and C=100.0 : 1.0000
```

```
#svm with linear kernel
linear_svc=SVC(kernel='linear', C=1.0)
linear_svc.fit(X_train,y_train)
y_pred_test=linear_svc.predict(X_test)
print('Accuracy score with linear kernel and C=1.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred_test)))
```

```
Accuracy score with linear kernel and C=1.0 : 1.0000
```

```
#svm with polynomial kernel
poly_svc=SVC(kernel='poly', C=1.0)
poly_svc.fit(X_train,y_train)
y_pred=poly_svc.predict(X_test)
print('Accuracy score with polynomial kernel and C=1.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Accuracy score with polynomial kernel and C=1.0 : 0.9630
```

```
#svm with polynomial kernel and C=100
poly_svc100=SVC(kernel='poly', C=100.0)
poly_svc100.fit(X_train,y_train)
y_pred=poly_svc100.predict(X_test)
print('Accuracy score with polynomial kernel and C=100.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Accuracy score with polynomial kernel and C=100.0 : 0.9815
```

```
#svm with sigmoid kernel
sigmoid_svc=SVC(kernel='sigmoid', C=1.0)
sigmoid_svc.fit(X_train,y_train)
y_pred=sigmoid_svc.predict(X_test)
print('Accuracy score with sigmoid kernel and C=1.0 : {0:0.4f}'. format(accuracy_score(y_test, y_pred)))
```

```
Accuracy score with sigmoid kernel and C=1.0 : 0.9815
```

Figure 7: SVM results with different kernels and 70/30 training/testing split

```
QSVC Score: 0.9814814814814815

Classification Report:
              precision    recall  f1-score   support

           1       1.00      0.95      0.97        19
           2       0.96      1.00      0.98        22
           3       1.00      1.00      1.00        13

    accuracy                           0.98        54
   macro avg       0.99      0.98      0.98        54
weighted avg       0.98      0.98      0.98        54
```

Figure 8. QSVM score with 70/30 training/testing split

```
X_train: (124, 13)
y_train: (124,)
X_test: (54, 13)
y_test: (54,)
Classification Report:
              precision    recall  f1-score   support

           0       0.95      1.00      0.97        19
           1       1.00      0.91      0.95        22
           2       0.93      1.00      0.96        13

    accuracy                           0.96        54
   macro avg       0.96      0.97      0.96        54
weighted avg       0.97      0.96      0.96        54
```

Figure 9. Multiclass NN with 70/30 training/testing split

Our findings revealed that the Radial Basis Function (RBF) and linear kernels for the SVM both produced 100% accurate results in both data training/testing splits. The polynomial kernel produced results with 94.4% accuracy with an increase to 97.2% when we increased our C value to 100 in our 80/20 split. The polynomial kernel produced improved results in our 70/30 split with 96.3% accuracy with an increase to 98.5% when C value increased to 100. The sigmoid kernel saw similar improvements. The QSVM produced an accuracy score of 97.2% in our 80/20 split, and then produced a 98.1% accuracy with our 70/30 split. The multiclass NN produced an accuracy score of 95% with our 80/20 split, and then a score of 97% in our 70/30 split. The QVC QNN produced an accuracy score of 95%.

To enhance interpretability, we visualized the models' performance using a confusion matrix. In the matrix, the y-axis represents the actual classes from the test dataset, while the x-axis corresponds to the predicted classes generated by the model. This visualization provides a clear overview of the classification performance and highlights areas for potential improvement.

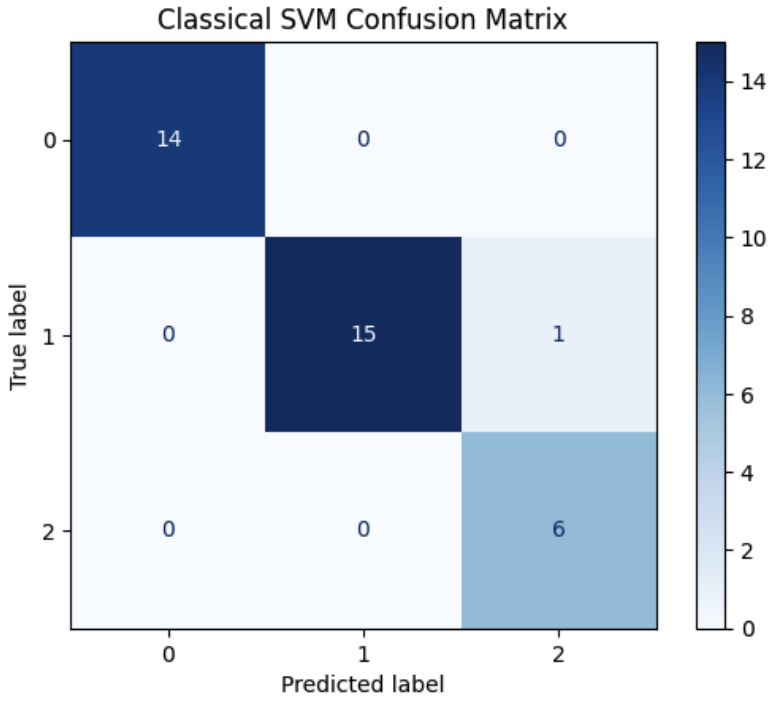

Figure 10. Classical SVM confusion matrix

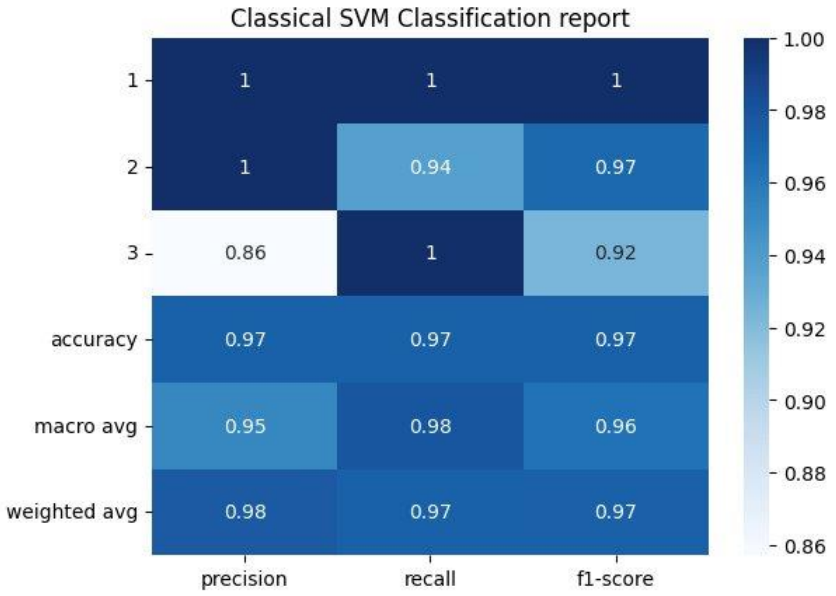

Figure 11. SVM Classification report

Quantum Support Vector Machine (QSVM)

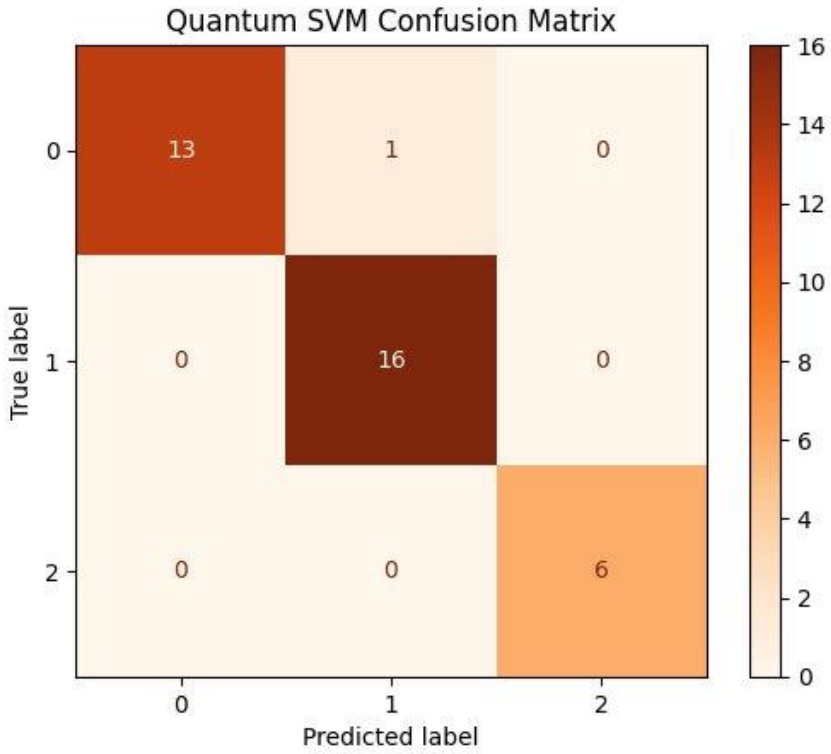

Figure 12. QSVM Confusion Matrix

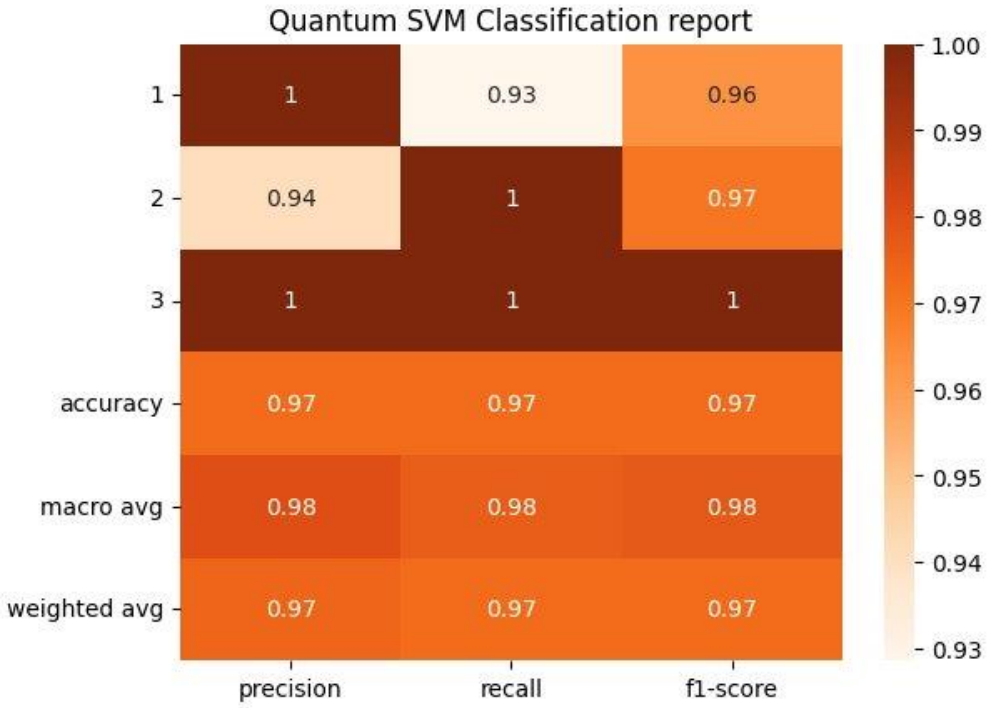

Figure 13. QSVM Classification report

Discussion

When comparing the performance of the quantum models to their classical counterparts, the accuracies and outcomes were remarkably similar. The confusion matrices and classification reports revealed slight variations in precision and recall between the two approaches, but the overall results were largely comparable. These findings suggest that while quantum machine learning does not yet surpass classical methods in terms of speed and precision, its performance indicates potential for future advancements as quantum technologies mature.

The choice of dataset was a critical factor in our analysis. The size and split ratio of the dataset significantly influenced the models' performance. For instance, a 70/30 split increased accuracy scores over the 80/20 split, while in our own experimentation, a 60/40 split resulted in a noticeable decrease in performance. These observations highlight the importance of dataset size and partitioning in determining model accuracy, further emphasizing the need for careful dataset preparation in machine learning applications.

As shown in the classification reports and the confusion matrices in Figures 10 - 13, the QSVM results closely resemble the results obtained from the SVM. To ensure these results closely mimic what we might observe on an actual quantum computer, we utilized

an error simulation tool called Aer. This tool simulated the quantum state of a system as a full state vector in Hilbert space, allowing us to model quantum behavior with high fidelity under ideal conditions. Although the AerSimulator is designed to be representative of the quantum computing environment, it is still a simulator producing results in an idealized environment, it is not a 1:1 replication of the unpredictable environment of utilizing actual quantum hardware.

For our quantum models we encountered many obstacles to make them run appropriately, the most significant problem we had was we could not transform the data to make it fit the quantum algorithm. Our first approach was to use amplitude encoding, but that produced a quantum circuit and the algorithm is waiting for data, not a quantum circuit. We later found the ZZFeatureMap, a Qiskit tool used specifically to do what we needed and map the features into a Hilbert space. Adjusting the settings to full entanglement produced better results.

The findings of this project underscore both the potential and limitations of quantum machine learning in its current state. While the quantum models performed comparably to their classical counterparts, they did not demonstrate a significant advantage in terms of accuracy or efficiency. In fact, the quantum model runtimes were exponentially larger than the classical ones. This suggests that quantum machine learning is still very new and requires further advancements in quantum hardware, algorithms, and error correction methods to fully realize its potential.

The impact of dataset size and split ratio on performance also raises important considerations for both classical and quantum models. Larger datasets with balanced splits appear to enhance model accuracy, but they also introduce greater computational demands. When we began this experiment, we had initially chosen a dataset size that had 16,000 samples in it. This caused our quantum models to time out. For quantum models, this challenge is compounded by the resource-intensive nature of quantum simulations, which are currently limited to idealized scenarios using tools like Qiskit Aer. These limitations highlight the importance of using actual quantum hardware for future studies to better understand the real-world applicability of quantum machine learning.

Future Work

For future work, we recommend utilizing a larger and more diverse dataset to investigate the impact of dataset size on model performance. This would help further validate or challenge the hypothesis that dataset size or linear separability among classes within data significantly influences the accuracy of both classical and quantum models.

Additionally, exploring datasets with varying levels of complexity could provide a broader understanding of how these models scale and adapt to different problem domains.

We also suggest expanding the scope of error simulation or, ideally, testing the quantum model on an actual quantum computer. While simulations like Qiskit Aer provide valuable insights, they operate under idealized conditions and may not fully capture the complexities of real quantum systems, such as noise and decoherence. Using actual quantum hardware would yield more realistic results and could uncover new challenges related to error correction and detection. This exploration could lead to novel approaches for addressing these issues, ultimately improving the practical viability of quantum machine learning in solving real-world problems.

Conclusion

In this study, we explored the integration of classical and quantum machine learning techniques for error classification tasks. By leveraging a dataset from the UC Irvine Machine Learning Repository, we focused on applying machine learning models, including a Support Vector Machine (SVM), a QSVM utilizing a quantum kernel, a multiclass neural network, and a variational quantum classifier quantum neural network. Our results demonstrated that the SVM, particularly with polynomial and sigmoid kernels, achieved ideal accuracy scores, providing a strong baseline for comparison. The performance of the quantum models, simulated using Qiskit's Aer error simulation, closely mirrored the classical models' results, with slight variations in precision and recall.

While the quantum models did not outperform the classical ones in terms of speed and precision, the similarities in performance suggest that quantum machine learning holds significant potential for error classification tasks. The use of quantum error simulations provided valuable insights into the behavior of quantum algorithms under idealized conditions, hinting at the possibility of achieving even better results with actual quantum hardware in the future. As quantum computing technology evolves, further experimentation with diverse datasets and quantum systems may unlock new opportunities for improving classification accuracy and computational efficiency in machine learning applications.

Repository: https://github.com/DarrylChajon/QuantumMachineLearningProject

References

"API Reference — Matplotlib 3.5.0 Documentation." *Matplotlib.org*, matplotlib.org/stable/api/index. Accessed Dec. 2024.

Bowles, Joseph, et al. "Better than Classical? The Subtle Art of Benchmarking Quantum Machine Learning Models.", Arxiv, 15 Mar. 2024. arxiv.org/pdf/2403.07059. Accessed Dec. 2024.

Havenstein, Christopher, et al. "Comparisons of Performance between Quantum and Classical Machine Learning." *SMU Data Science Review*, vol. 1, no. 4, 2018, p. 11, scholar.smu.edu/cgi/viewcontent.cgi?article=1047&context=datasciencereview. Accessed Dec. 2024.

Hidalgo, Matheus Cammarosano. "A Simple Introduction to Quantum Enhanced SVM - towards Data Science." *Medium*, Towards Data Science, 3 Oct. 2023, towardsdatascience.com/a-simple-introduction-to-quantum-enhanced-svm-bee893a4377c. Accessed Dec. 2024.

Khan, Tariq M., and Antonio Robles-Kelly. "Machine Learning: Quantum vs Classical." *IEEE Access*, vol. 8, no. 8, 2020, pp. 219275–219294, dro.deakin.edu.au/eserv/DU:30146397/khan-machinelearningquantumvs-2020.pdf, https://doi.org/10.1109/access.2020.3041719. Accessed Dec. 2024.

Nolancmas. "Binary Classification Using Quantum Machine Learning." *Medium*, 8 Apr. 2023, medium.com/@nolancmas/quantum-machine-learning-classification-quantum-support-vector-machines-qsvm-vs-quantum-neural-a597c41f2f16. Accessed Dec. 2024.

NumPy. "Overview — NumPy V1.19 Manual." *Numpy.org*, 2022, numpy.org/doc/stable/. Accessed Dec. 2024.

"Qiskit Machine Learning 0.8.1." *Github.io*, Github, 2018, qiskit-community.github.io/qiskit-machine-learning/. Accessed Dec. 2024.

Scikit-learn. “Scikit-Learn: Machine Learning in Python.” *Scikit-Learn.org*, scikit-learn.org/stable/. Accessed Dec. 2024.

seaborn. “Seaborn: Statistical Data Visualization — Seaborn 0.9.0 Documentation.” *Pydata.org*, 2012, seaborn.pydata.org/. Accessed Dec. 2024.

“Ucimlrepo Package.” *GitHub*, 18 Oct. 2023, github.com/uci-ml-repo/ucimlrepo. Accessed Dec. 2024.

Veloso, Thiago. “Quantum Support Vector Machine (QSVM) - MIT 6.S089—Intro to Quantum Computing - Medium.” *Medium*, MIT 6.s089—Intro to Quantum Computing, 30 Jan. 2022, medium.com/mit-6-s089-intro-to-quantum-computing/quantum-support-vector-machine-qsvm-134eff6c9d3b. Accessed Dec. 2024.

Zhao, Yue. “Benchmarking Machine Learning Models for Quantum Error Correction.” *Arxiv.org*, 2024, arxiv.org/html/2311.11167v3#S2.F2. Accessed Dec. 2024.