Kernel Alignment for Quantum Support Vector Machines Using Genetic Algorithms

Floyd M. Creevey,[1,*] Jamie A. Heredge,[1,†] Martin E. Sevior,[1,‡] and Lloyd C. L. Hollenberg[1,§]

[1] *School of Physics, University of Melbourne, VIC, Parkville, 3010, Australia.*

(Dated: December 5, 2023)

The data encoding circuits used in quantum support vector machine (QSVM) kernels play a crucial role in their classification accuracy. However, manually designing these circuits poses significant challenges in terms of time and performance. To address this, we leverage the GASP (Genetic Algorithm for State Preparation) framework for gate sequence selection in QSVM kernel circuits. We explore supervised and unsupervised kernel loss functions' impact on encoding circuit optimisation and evaluate them on diverse datasets for binary and multiple-class scenarios. Benchmarking against classical and quantum kernels reveals GA-generated circuits matching or surpassing standard techniques. We analyse the relationship between test accuracy and quantum kernel entropy, with results indicating a positive correlation. Our automated framework reduces trial and error, and enables improved QSVM based machine learning performance for finance, healthcare, and materials science applications.

Keywords: quantum computing, genetic algorithm, quantum machine learning, SVM, QSVM

I. INTRODUCTION

Support vector machines (SVMs) are a popular class of classical machine learning algorithms that are widely used for classification and regression tasks in various fields, such as finance [1], healthcare [2, 3], and chemistry [4]. They work by finding the hyperplane that best separates the given data. Recently, quantum SVMs (QSVMs) have been introduced as the generalisation of SVMs that use a quantum kernel function to measure the similarity between data points [5]. The quantum kernel function is typically implemented using a quantum circuit that encodes the features of the data points into the amplitudes of a quantum state, and then applies a set of quantum gates to compute the inner product between the quantum states. One advantage of QSVMs over classical SVMs is their ability to construct kernels that are not classically simulable [6]. Another advantage is their potential for enhanced performance on certain tasks, such as the classification of non-linearly separable data, due to the ability of the quantum kernel function to operate in a higher-dimensional feature space than classical SVMs [7]. These advantages make QSVMs an attractive area of research for improving the performance of classical machine learning algorithms. However, designing an effective quantum kernel function is a non-trivial task, and has been the subject of much research in the field of quantum machine learning [8].

There are several techniques that are currently used to design quantum circuits for machine learning tasks, including analytical methods [9], numerical [10], and variational methods [11, 12]. Each of these techniques has its own strengths and weaknesses, with the choice of technique depending on the specific problem and the available resources. Analytical methods are based on mathematical analysis and aim to find the optimal circuit design that minimises the error in the output. These methods can be computationally efficient, but they may not be suitable for all problems, as they rely on the availability of exact mathematical solutions. Numerical optimisation techniques involve the use of optimisation algorithms to find the best circuit design by minimising a cost function that measures the error in the output. These methods are generally more flexible than analytical methods, but they can be computationally intensive and may require large amounts of data. Variational methods employ parameterised quantum circuits, where the parameters are optimised to minimise output error, based on the idea of representing the quantum state through a neural network and optimising network parameters using classical algorithms. Such methods are highly flexible and can be used for a wide range of problems, but they may not always produce the most accurate results.

The approach toward optimising the quantum kernel function described in this paper is to use a genetic algorithm, building from the genetic algorithm for state preparation (GASP) framework presented in [13]. Previous work has been done on the generation of optimal ad-hoc kernel function quantum circuits for classification by using a quantum support vector machine [8][14], presenting the use of a genetic algorithm and comparison with classical classifiers. This paper will examine different distinct fitness functions for the generation of the kernel function and analyse performance based on both their classification accuracy and entropy of entanglement. A genetic algorithm is a heuristic search algorithm that mimics the process of natural selection [15]. The genetic algorithm can be used to search for an optimal quantum circuit that produces an accurate kernel matrix for the SVM while taking into account various constraints such as the number of qubits, the gate depth, and the connectivity of the quantum circuit. We explore the use of a genetic algorithm to optimise the kernel function quantum circuit of a QSVM on various datasets. To generate the quantum circuits, we use a set of four gates: the single-qubit X, $\sqrt{X}$, and R_z gates, and the two-qubit CX gate. These form a universal gate set for quantum computation and are commonly used in quantum algorithms

* fcreevey@student.unimelb.edu.au
† heredgej@student.unimelb.edu.au
‡ martines@unimelb.edu.au
§ lloydch@unimelb.edu.au

[16]. We evaluate the performance of the kernel function generated by the genetic algorithm using the accuracy of classification results on the testing set and compare it with classical kernels and quantum kernels.

Our results show that the kernel function quantum circuits generated by the genetic algorithm using this gate set perform comparably or better than classical kernels [17], and consistently outperform the standard PauliZZ quantum kernel [18]. This approach additionally aligns kernel circuits in a manner that enables them to traverse the Hilbert space region where the solution classification is situated. These results demonstrate the potential of QSVMs, indicating that employing a genetic algorithm for quantum circuit optimisation offers an alternative to manually designing quantum circuits and can be extended to larger and more complex datasets.

The remainder of this paper will have the following structure. Section II will give a brief summary of SVMs, QSVMs and genetic algorithms. Section III will describe our proposed method for kernel generation in detail. Section IV will present the results, and section V our conclusions and potential future work.

II. BACKGROUND

Genetic algorithms (GAs) are a class of optimisation algorithms inspired by the process of natural selection in biology. The basic idea behind GAs is to mimic the evolution process of living organisms by iteratively generating and refining candidate solutions to an optimisation problem. Each candidate solution is referred to as an 'individual' in the GA context, and a collection of individuals forms a 'population'. GAs operate by iteratively applying three key operations to the population: selection, crossover, and mutation. Selection involves choosing the fittest individuals from the population to be used as parents for the next generation. Crossover involves recombining the genetic material of two parents to create new offspring. Finally, mutation involves randomly changing some of the genetic material in the offspring to introduce variation. The genetic material in GAs is typically represented as a string of binary digits (i.e. 0s and 1s), called a 'chromosome'. Each element of the chromosome is referred to as a 'gene', which encodes a specific aspect of the solution to the optimisation problem. For example, in a GA that is being used to optimise the coefficients of a polynomial function, each gene might represent a particular coefficient. The fitness function is a crucial component of a GA, as it determines the objective measure of quality for each individual in the population. The fitness function maps each individual to a real number that quantifies how well the individual solves the optimisation problem. The fitness function is typically problem-specific and is designed to reflect the optimisation goals of the problem at hand. The process of generating a new generation of individuals in a GA involves the following steps:

1. Selection: Choose the fittest individuals from the population to be used as parents for the next generation. The probability of an individual being selected is proportional to its fitness score.
2. Crossover: Create new offspring by combining the genetic material of two parents. This is done by selecting a crossover point on the chromosome and swapping the genes to the right of the crossover point between the two parents.
3. Mutation: Introduce variation into the population by randomly changing some of the genes in the offspring. This is typically done by flipping some of the bits in the chromosome.

The process of generating new generations continues until a stopping criterion is met, such as a maximum number of generations or a satisfactory level of fitness. The effectiveness of GAs depends on a number of factors, including the choice of representation for the genetic material, the design of the fitness function, and the selection, crossover, and mutation operators. With the right combination of these factors, GAs can be effective at finding high-quality solutions to a wide range of optimisation problems, including those that are difficult for traditional optimisation methods to solve.

Support Vector Machines (SVMs) are a popular supervised machine learning method for classification and regression tasks. SVMs aim to find a hyperplane that separates the data into two or more classes, with the largest possible margin between the closest data points to the hyperplane. These closest data points are called support vectors, and the margin is defined as the perpendicular distance between the hyperplane and the closest support vectors. The intuition behind this approach is that the larger the margin, the more robust the classifier will be to new data. Consider a dataset $\mathcal{D}$ of n datapoints each with m features, of the form $(\mathcal{X}, \vec{y})$, where,

$$\mathcal{X} = \begin{pmatrix} \vec{X}_1 \\ \vec{X}_2 \\ \vdots \\ \vec{X}_n \end{pmatrix} = \begin{pmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,m} \\ x_{2,1} & x_{2,2} & \dots & x_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \dots & x_{n,m} \end{pmatrix}, \tag{1}$$

and,

$$\vec{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}. \tag{2}$$

Here we review the background on SVMs [19]. The first step is to define the SVM Lagrangian, also called the primal problem,

$$\begin{aligned} \mathcal{L}(\vec{w}, b, \vec{\alpha}) &= \frac{1}{2}||\vec{w}||_2^2 - \sum_{i=1}^{n} \alpha_i (y_i(\vec{X}_i^T \cdot \vec{w} + b) - 1) \\ &= \frac{1}{2}\vec{w}^T \cdot \vec{w} - \sum_{i=1}^{n} \alpha_i y_i \vec{X}_i^T \cdot \vec{w} \\ &\quad + b\sum_{i=1}^{n} \alpha_i y_i + \sum_{i=1}^{n} \alpha_i, \end{aligned} \tag{3}$$

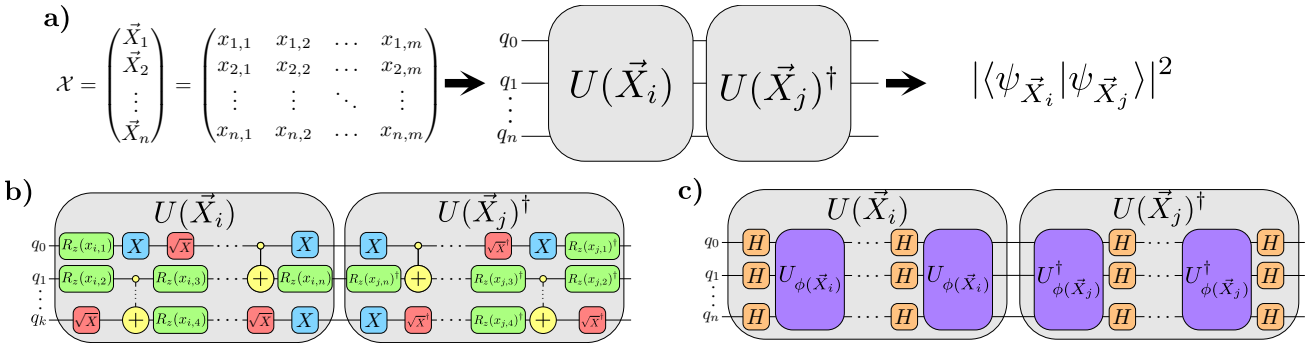

Figure 1. Basic overview of QSVMs. **a)** The datapoints, $\mathcal{X}$, with labels $\vec{y}$, are used to construct the elements of the kernel matrix, defined as the piecewise inner product, for the QSVM using the quantum kernel function for the given circuit structure. **b)** An example circuit that could be produced by the GA method. The GA is stochastic by nature, the number of features used and the number of qubits in the circuit will vary, as the circuit structure will likely be different each time a circuit is generated. In the GA circuit the X and $\sqrt{X}$ gates represent the Pauli X and the square root of the Pauli X respectively, not to be confused with the input data $\vec{X}_i$. **c)** Circuit structure of the PauliZZ quantum kernel function where H symbolises the Hadamard gate, and $U_{\phi(\vec{X}_i)} = \otimes_{i=1}^{n} U(x_{i,j}) \prod_{(i,j)\in E} CZ(i,j)$.

where $\vec{w}$ are the weights, b is the bias, and $\vec{\alpha}$ are the Lagrange multipliers. Then compute the partial derivatives with respect to its primal variables,

$$\frac{\partial \mathcal{L}}{\partial \vec{w}} = 0_d \rightarrow \vec{w}^* = \sum_{i=1}^{n} \alpha_i y_i \vec{X}_i, \tag{4}$$

and,

$$\frac{\partial \mathcal{L}}{\partial b} = 0 \rightarrow 0 = \sum_{i=1}^{n} \alpha_i y_i. \tag{5}$$

With this, we can solve the dual problem,

$$\alpha_1^*, \alpha_2^*, \ldots, \alpha_n^* = \max_{\alpha_1,\alpha_2,\ldots,\alpha_n} \mathcal{L}(\vec{w}^*, b^*, \vec{\alpha}), \tag{6}$$

subject to $\alpha_i \geq 0$, $i \in 1, 2, \ldots, n$ and $\sum_{i=1}^{n} \alpha_i y_i = 0$, and with solutions (1) and (2) we find,

$$\mathcal{L}(\vec{w}^*, b^*, \vec{\alpha}) = -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \vec{X}_i^T \cdot \vec{X}_j + \sum_{i=1}^{n} \alpha_i, \tag{7}$$

and the Lagrange multipliers α_i can be found, and used to find $\vec{w}^*$,

$$\vec{w}^* = \sum_{i=1}^{n} \alpha_i y_i \vec{X}_i. \tag{8}$$

With $\vec{w}^*$ we can find the margin width $l = 2/||\vec{w}^*||$, and b^*, which can be computed from any support vector, $y_i = \vec{X}_i^T \cdot \vec{w}^* + b^*$.

For a non-linear classification, first a good feature transformation ϕ must be found to map the datapoints into linearly separable sets,

$$\begin{aligned} \mathcal{L}(\vec{w}^*, b^*, \vec{\alpha}) = &\sum_{i=1}^{n} \alpha_i \\ &- \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \phi(\vec{X}_i)^T \phi(\vec{X}_j), \end{aligned} \tag{9}$$

where $\phi(\vec{X}_i)^T \cdot \phi(\vec{X}_j)$ is a similarity measure between the transformed datapoints. We can then identify the kernel function $\mathcal{K}(\vec{X}_i, \vec{X}_j) = \phi(\vec{X}_i)^T \phi(\vec{X}_j)$, which defines the inner products in the transformed space.

For multi-class classification the key strategy is to create a two-class classifier that distinguishes samples in a particular class c from all other classes, i.e label points in class c as $(+1)$ and points not in class c as (-1). Then solve for all C classes,

$$\begin{aligned} \vec{X}^T \cdot \vec{w}_c^* + b_c^* = 0 \\ c = 1, 2, \ldots, C, \end{aligned} \tag{10}$$

so that all points from class c will lie on the positive side of its decision boundary (w_c^*, b_c^*), while points from other classes lie on its negative side. Hence, datapoint $\vec{X}_i$ belongs to class c if it satisfies the two following inequalities,

$$\begin{aligned} &\vec{X}_i^T \cdot \vec{w}_c^* + b_C^* > 0 \\ &\vec{X}_i^T \cdot w_j^* + b_j^* < 0 \text{ for } j = 1, 2, \ldots, c, \text{ and } j \neq c. \end{aligned} \tag{11}$$

However this is typically not a good approach, as it does not allow samples in ambiguous space to be assigned labels [20]. A solution to this is to use the fusion rule. The fusion rule generalises to assign a label for each sample $\vec{X}_i$ by finding not the classifier that produces a positive evaluation $\vec{X}_i^T \cdot \vec{w}_c^* + b_c^* > 0$, but by assigning $\vec{X}_i$ the class label c with the largest evaluation (even when negative),

$$y = \max_{c=1,2,\ldots,C} \vec{X}_i^T \cdot w_c^* + b_c^*, \tag{12}$$

this assigns labels to the entire space and effectively handles overlapping classes. Using the fusion rule, C individual classifiers are learned, each distinguishing one class from the remainder of the data. The learned classifiers are then combined to make final assignments.

Quantum Support Vector Machines (QSVMs) are a quantum computing-based variant of SVMs, designed for classification and regression tasks. QSVMs leverage the principles of quantum computing to potentially offer advantages in handling high-dimensional data and solving complex optimisation problems. Consider the same dataset $\mathcal{D}$ as above. In QSVMs, input data points are encoded as quantum states. These quantum states are represented in Dirac notation, i.e $|\psi_{\vec{X}}\rangle$. Each data point $\vec{X}$ is mapped to a quantum state $|\psi_{\vec{X}}\rangle$ as,

$$|\psi_{\vec{X}}\rangle = U(\vec{X})|0\rangle, \tag{13}$$

where U represents a quantum transformation or operator. The central element of QSVMs is the quantum kernel function, which quantifies the similarity between quantum states. The quantum kernel function is defined as,

$$\mathcal{K}(\vec{X}_i, \vec{X}_j) = |\langle\psi_{\vec{X}_i}|\psi_{\vec{X}_j}\rangle|^2, \tag{14}$$

i.e. the inner product of the quantum states $|\psi_{\vec{X}_i}\rangle$ and $|\psi_{\vec{X}_j}\rangle$ (see Figure 1a). This is analogous to the kernel trick in classical SVMs but operates in a quantum state space.

A commonly used quantum kernel is repeated layers of the PauliZZ feature map [5]. It can be constructed as,

$$W(\vec{X}_i) = H^{\otimes n} U_{\phi(\vec{X}_i)}, \tag{15}$$

where $U_{\phi(\vec{X}_i)} = \otimes_{i=1}^{n} U(x_{i,j}) \prod_{(i,j)\in E} CZ(i,j)$. A basic overview of QSVMs is displayed in Figure 1.

Eigenvalue Analysis of Kernel Matrices assesses the fitness of a kernel matrix, $\mathcal{K}$, created with the corresponding kernel circuit, and optimise by maximising its largest normalised eigenvalue. Let $\lambda_1, \lambda_2, \ldots, \lambda_n$ be the eigenvalues of $\mathcal{K}$, where n is the dimension of $\mathcal{K}$, the maximum normalised eigenvalue of is $\frac{\lambda_1}{n}$. This method is motivated by Spectral Clustering [21], and is similar to kernel principal component analysis (KPCA) [22]. Maximising the largest normalised eigenvalue can lead to a focus on the dominant mode in the data, potentially resulting in dimensionality reduction.

Entropy of Entanglement of a kernel circuit is calculated with the Von Neumann Entropy as,

$$S = \frac{\sum_{i=1}^{n} -\mathrm{Tr}(\rho_\mathrm{i} \ln \rho_\mathrm{i})}{n} \tag{16}$$

where n is the number of qubits in the circuit, and ρ_i is the partial trace of the kernel matrix with respect to the ith qubit.

III. GENETIC ALGORITHM QSVM

In this paper, we first perform classification on several synthetic datasets, moons, XOR, and circles, before performing classification on the real datasets, Iris, Wine, and Breast Cancer, all from scikitlearn [23]. We then perform classification on the more challenging real datasets of Irrigation [24], Drug Classification [25], and Parkinson's [26]. The datasets are all initially analysed with principal component analysis (PCA) [27] to determine the number of components required to express 95% of the variance in the data, and then used to reduce the number of components to that size. With this processed data, our method then uses a genetic algorithm similar to that of GASP [13], to optimise the kernel circuit for the QSVM. We start with an initial individual, which is a quantum circuit composed of gates from the gate set $\mathcal{G} = X, \sqrt{X}, R_z, CNOT$. The genetic algorithm then creates k mutated individuals by creating k copies of the individual, then mutating each individual with a probability m_p, set to 50% for these results. For the supervised GA technique, each individual's fitness is assessed as the classification accuracy of running the QSVM with the corresponding kernel circuit on the training data. For the unsupervised GA technique, each individual's fitness is assessed by eigenvalue analysis of the kernel matrix. It is important to note here that the terms 'Supervised' and 'Unsupervised' refer to the technique for optimisation of the kernel circuit by genetic algorithm. The 'Supervised' technique is supervised as it requires data labels in the assessment of the kernel circuit classification accuracy for its fitness. The 'Unsupervised' technique does not use labels in the fitness assessment of the kernel circuit; the fitness is assessed with only the kernel matrix, as the maximisation of the largest normalised eigenvalue. However, the overall methodology of training support vector machines, whether they are classical or quantum, is always a supervised task. The individual with the highest training fitness is then selected and their validation fitness is assessed in the same manner the training fitness was assessed, except validation data is used instead of training data. If the validation fitness of the most fit mutated individual is greater than the validation fitness of the current individual, it becomes the new current individual. We repeat this process for a fixed number of iterations, where each iteration is called a 'generation', or until the fitness of the individual reaches a desired threshold. if the desired threshold fitness is not achieved within the number of generations given, the number of genes is increased or decreased according to a binary search. This ensures that the individuals produced have the lowest amount of genes required to achieve the desired

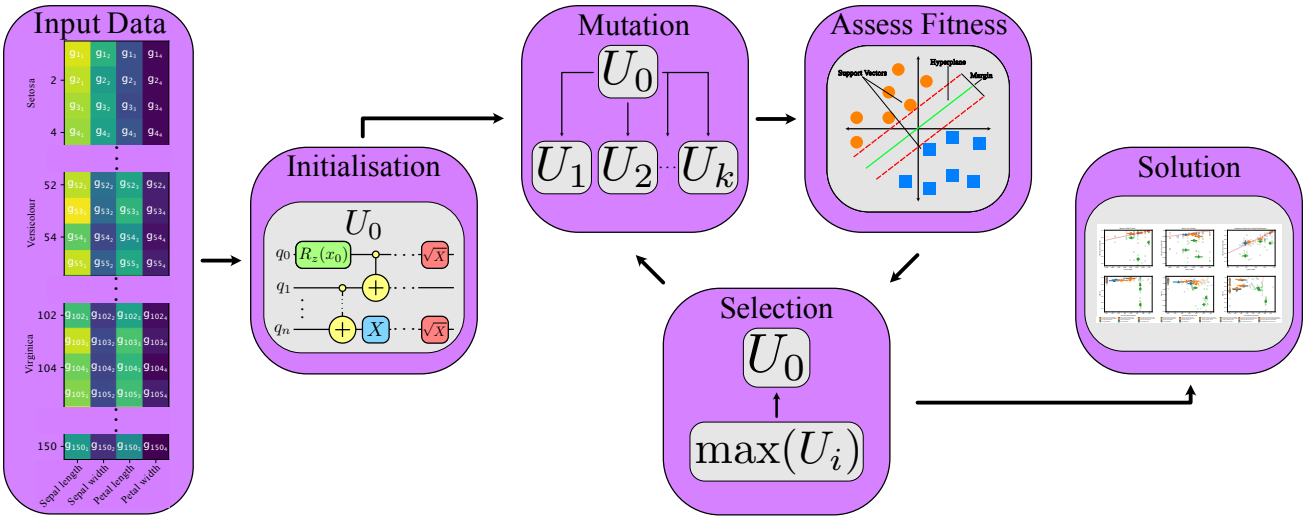

Figure 2. Overview of the GA for QSVM approach, given a dataset. First, create an initial individual U_0 which is a quantum kernel circuit. Then mutate the individual with probability $p = 50\%$ k times, to produce a k size population of individuals. Then assess the fitness of the quantum kernel circuits determined by each individual in the population: if supervised, the fitness is the result of training a QSVM using the given kernel, if unsupervised, the fitness is the maximum normalised eigenvalue of the kernel matrix of the circuit. Then select the most fit individual in the population, if it is more fit than U_0 it becomes U_0. Repeat until the desired fitness is achieved.

fitness. A flow chart of this methodology can be seen in Figure 2, and pseudo-code for this methodology is shown in Appendix A Algorithm 1. Overall, our methods optimise the kernel function for the QSVM using a genetic algorithm with either a supervised or an unsupervised technique. We demonstrate the effectiveness of our method on nine benchmark datasets.

IV. RESULTS

The methods were tested on the Moons, XOR, Circles, Wine, Iris, Cancer, Irrigation, Parkinson's, and Drug Classification data sets. The Moons, XOR, and circles datasets were synthetically generated with 400 samples each. The Wine, Iris, Cancer, Irrigation, Parkinson's and Drug Classification datasets were all real data, and all data points were used. PCA was used to determine the number of features required for each dataset to explain 95% of the variance in the data. The results of this analysis can be seen in Figure 3, and the number of features used for each dataset can be seen in Table I. Each data set was split into 20% testing data, 48% training data, and 32% validation data. The total number of data points and split of data for each dataset is displayed in Appendix B Table II. The data was then scaled between $[-\pi/2, \pi/2]$, and tested 10 times for each method to determine the prediction test accuracy and entropy of entanglement of each kernel. The results are displayed in Figure 4.

The results show that over the ten tests for each kernel, both the supervised and unsupervised GA generated kernels consistently outperform the PauliZZ kernel, and perform comparably to the RBF kernel, on each dataset. When viewing the decision boundaries produced by each technique, examples for Moons, XOR, Circles, and Irrigation data are shown in Figure 5, it becomes more clear why certain techniques achieve higher accuracies than others. For instance, it can be seen that the GA and RBF kernels have smooth, defined decision boundaries, clearly separating the two classes. In contrast, the PauliZZ produces a patchy decision boundary that does not appropriately separate the classes. Over the ten tests, the highest test accuracy GA kernel is always equal to or higher than the highest test accuracy RBF kernel, however, there is more variance in the test accuracy achieved. This is not unexpected, as the GA is a stochastic process, and in real-world applications, the best of several tests would be chosen. In general, the supervised and unsupervised GA techniques seem to perform comparably, which implies, for these datasets, this technique does not require the supervised training of QSVMs as a fitness metric to develop kernels. It is also interesting to note that the unsupervised GA technique seems to perform better in real datasets for the datasets analysed. This is a very interesting result and should be the subject of further research, as this seems to suggest that, for the datasets analysed, there are underlying features present in real data that allow the unsupervised technique to outperform the supervised technique , which are not present in synthetic data. These features could be taken advantage of in machine learning. When viewing the entropy plots, it can be seen that both the GA methods produce kernels of higher test accuracy than the PauliZZ kernel, with entanglement being fairly independent of the dataset, whereas the PauliZZ kernel seems to have a large amount of variance in entanglement between datasets. Further, when a linear regression is applied to the test accuracy and entropy data, it can be seen that there is a positive gradient, as seen in Figure 6, implying that as the entropy of a kernel circuit increases, the classification accuracy of the corresponding QSVM increases.

By selecting the optimal kernel from the population based on training fitness, and then only updating the base individual if the validation accuracy was higher, the GA is forced to maintain generality while

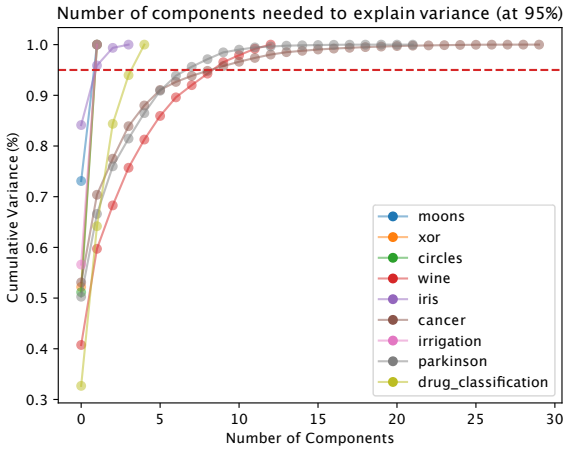

Figure 3. Comparison of the number of features required to explain 95% variance in the dataset for Moons, XOR, Circles, Wine, Iris, Cancer, Irrigation, Parkinson's, and Drug Classification datasets, coloured blue, orange, green, red, purple, brown, pink, grey, and light green, respectively. The dots represent features, ordered from the feature that explains the highest amount of variance in the data, to the feature that explains the lowest amount of variance in the data, the red dashed line represents 95% of the data being explained.

Table I. **Number of components needed to explain 95% variance in each dataset from analysis with PCA.**

Dataset	Total Number of Components	95% Variance Number of Components
Moons	2	2
XOR	2	2
Circles	2	2
Wine	13	10
Iris	4	2
Cancer	30	10
Irrigation	2	2
Parkinson's	22	8
Drug Classification	5	5

increasing accuracy. Our results demonstrate that a genetic algorithm can be used to optimise QSVM kernels, producing circuits that outperform manually designed classical and quantum kernels on standard classification tasks. The genetic algorithm appears to be learning to encode more complex features in the data and to use entanglement more efficiently.

V. CONCLUSION

In this paper, we have presented an approach for optimising QSVM kernels using a genetic algorithm, based on GASP [13]. Our approach has been shown to outperform manually designed kernels on standard toy datasets, demonstrating the potential of this technique for improving the performance of QSVMs. The results show that the genetic algorithm is able to learn to encode more complex features in the data and to use entanglement more efficiently. This suggests that our approach may be useful for identifying patterns in complex datasets, especially those that are difficult to analyse with classical machine learning techniques. This efficiency, combined with improved performance, makes our approach a promising tool for researchers and practitioners in various fields, such as finance, healthcare, and materials science, where data analysis and prediction are crucial. There are several avenues for future research in this area. One potential direction is to investigate the effectiveness of our approach on larger and more complex datasets, such as those in real-world applications. Another interesting area for exploration is the use of other genetic algorithms or optimisation techniques, such as simulated annealing or particle swarm optimisation, to further improve the performance of QSVMs. It would also be valuable to investigate the possibility of underlying features present in real data that allow the unsupervised technique to outperform the supervised technique. In conclusion, our study provides an effective approach for optimising QSVM kernel circuits using a genetic algorithm. The results suggest that this technique has significant potential for improving the performance of QSVMs on standard classification tasks, as well as for identifying patterns in complex data in various fields.

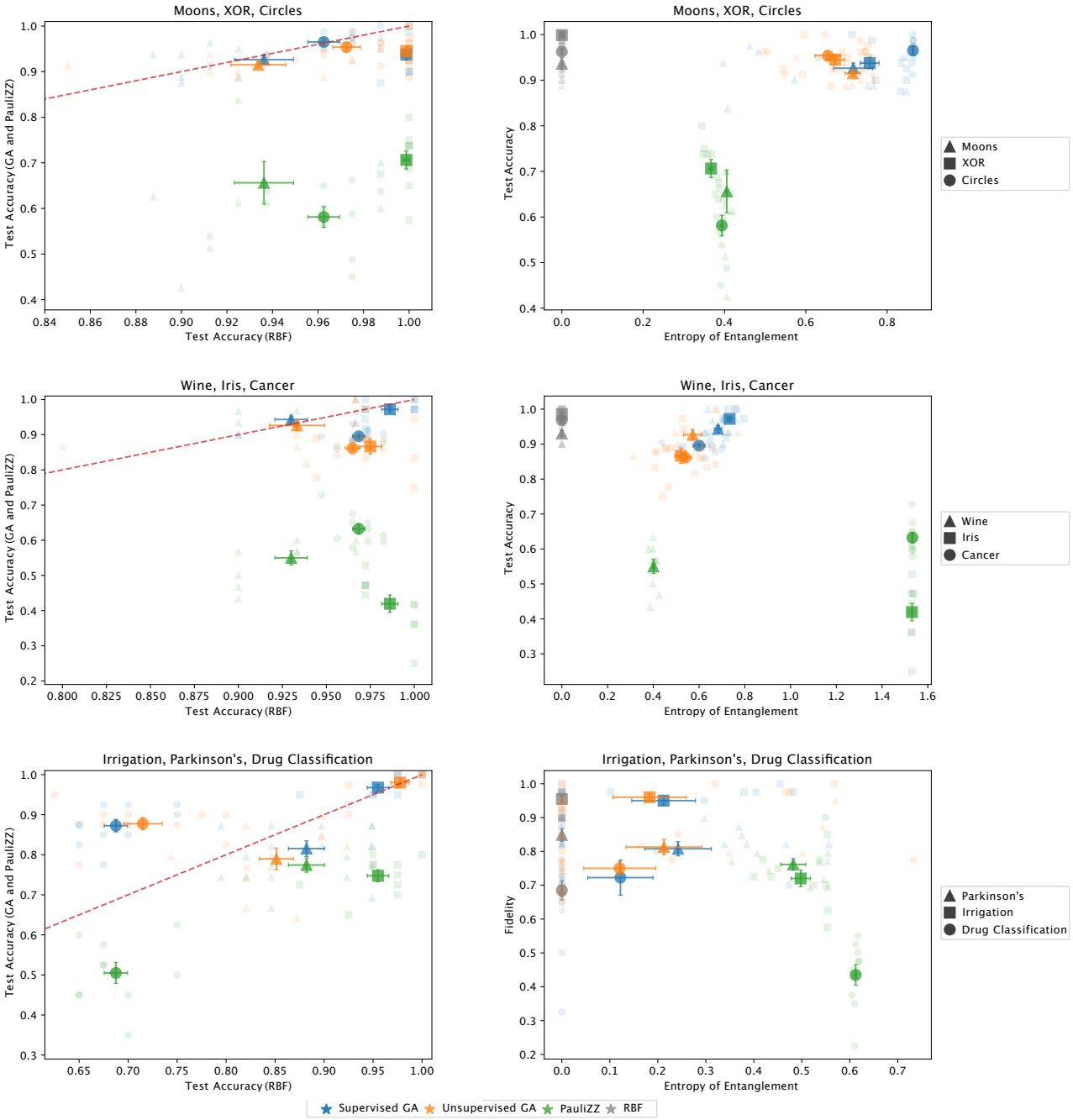

Figure 4. Comparison of supervised GA, unsupervised GA, PauliZZ, and RBF generated kernels on Moons, XOR, Circles, Wine, Iris, Cancer, Irrigation, Parkinson's, and Drug Classification datasets. The supervised GA, unsupervised GA, PauliZZ, and RBF kernels are blue, orange, green, and grey, respectively. The top three plots show the comparison of the fidelities achieved for the supervised GA, unsupervised GA, and PauliZZ kernels against the test accuracy achieved by the RBF kernel. The red dashed line represents where the test accuracy of the supervised GA, unsupervised GA or PauliZZ kernel would be equal to the test accuracy of the RBF kernel. The bottom three plots show the comparison of the test accuracy achieved by each kernel against the entropy of entanglement of the kernel. The large markers represent the average result over the ten tests conducted for each kernel on each dataset, with error bars included. The small markers represent each of the individual tests.

VI. ACKNOWLEDGEMENTS

This research was supported by the University of Melbourne through the establishment of the IBM Quantum Network Hub at the University. FMC and JAH are supported by Australian Government Research Training Program Scholarships. This research was supported by The University of Melbourne's Research Computing Services and the Petascale Campus Initiative.

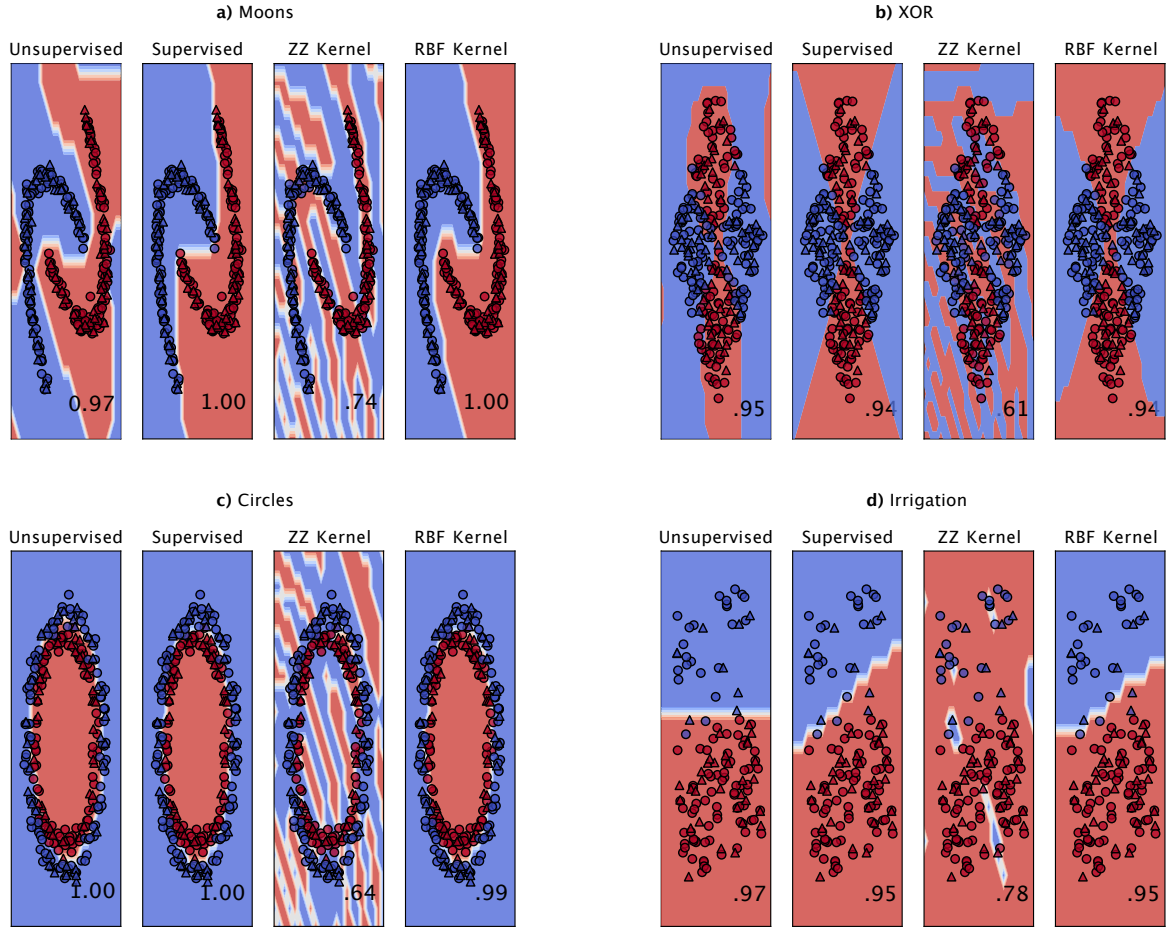

Figure 5. Decision boundaries for the Moons, XOR, Circles, and Irrigation datasets. Training data and testing data are represented by circles and triangles, respectively. The decision bounds vary from red to blue, with darker shades being higher confidence and white being the region of least confidence.

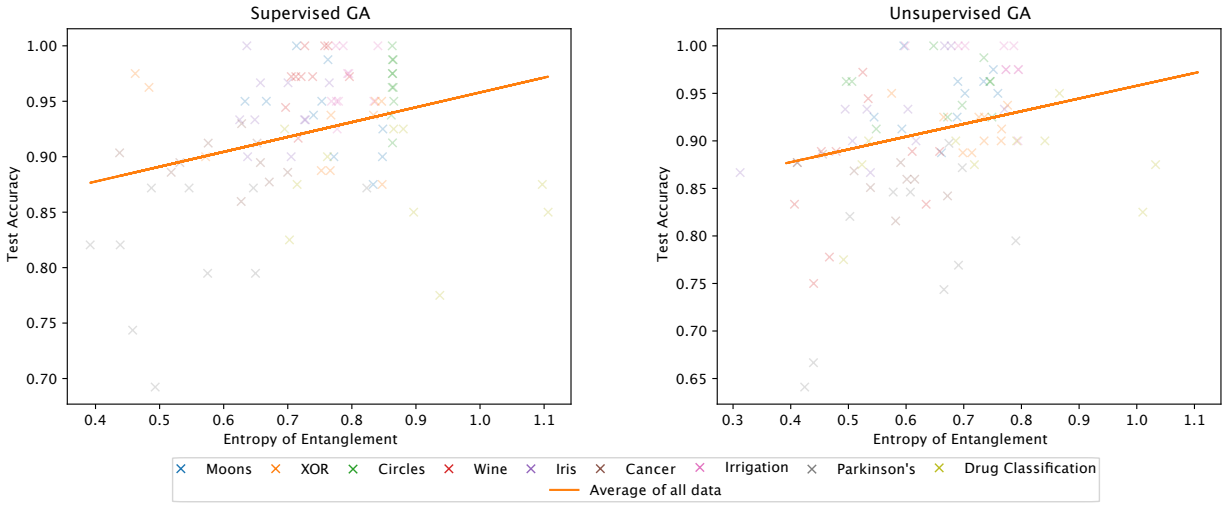

Figure 6. Comparison of supervised and unsupervised GA kernel entropies compared to test accuracy of the given kernel on Moons, XOR, Circles, Wine, Iris, Cancer, Irrigation, Parkinson's, and Drug Classification datasets, coloured blue, orange, green, red, purple, brown, pink, grey, and light green, respectively. The crosses represent individual tests.

[1] A. Kurani, P. Doshi, A. Vakharia, and M. Shah, Annals of Data Science **10**, 183 (2023).

[2] W. Yu, T. Liu, R. Valdez, M. Gwinn, and M. J. Khoury, BMC Medical Informatics and Decision Making **10**, 16 (2010).

[3] C. Venkatesan, P. Karthigaikumar, A. Paul, S. Satheeskumaran, and R. Kumar, IEEE Access **6**, 9767 (2018), conference Name: IEEE Access.

[4] M. Guangli and C. Yiyu, (2006).

[5] V. Havlíček, A. D. Córcoles, K. Temme, A. W. Harrow, A. Kandala, J. M. Chow, and J. M. Gambetta, Nature **567**, 209 (2019), number: 7747 Publisher: Nature Publishing Group.

[6] P. Rebentrost, M. Mohseni, and S. Lloyd, Physical Review Letters **113**, 130503 (2014), publisher: American Physical Society.

[7] W. S. Noble, Nature Biotechnology **24**, 1565 (2006), number: 12 Publisher: Nature Publishing Group.

[8] S. Altares-López, A. Ribeiro, and J. J. García-Ripoll, Quantum Science and Technology **6**, 045015 (2021), publisher: IOP Publishing.

[9] M. Benedetti, E. Lloyd, S. Sack, and M. Fiorentini, Quantum Science and Technology **4**, 043001 (2019), publisher: IOP Publishing.

[10] Y. Suzuki, H. Yano, Q. Gao, S. Uno, T. Tanaka, M. Akiyama, and N. Yamamoto, Quantum Machine Intelligence **2**, 9 (2020).

[11] P. Wang, M. Usman, U. Parampalli, L. C. L. Hollenberg, and C. R. Myers, IEEE Transactions on Quantum Engineering **4**, 1 (2023), conference Name: IEEE Transactions on Quantum Engineering.

[12] K. Nakaji, S. Uno, Y. Suzuki, R. Raymond, T. Onodera, T. Tanaka, H. Tezuka, N. Mitsuda, and N. Yamamoto, Physical Review Research **4**, 023136 (2022).

[13] F. M. Creevey, C. D. Hill, and L. C. L. Hollenberg, Scientific Reports **13**, 11956 (2023), number: 1 Publisher: Nature Publishing Group.

[14] S. Altares-López, J. J. García-Ripoll, and A. Ribeiro, "AutoQML: Automatic Generation and Training of Robust Quantum-Inspired Classifiers by Using Genetic Algorithms on Grayscale Images," (2022), arXiv:2208.13246 [quant-ph].

[15] M. Mitchell, *An Introduction to Genetic Algorithms*, Complex Adaptive Systems (A Bradford Book, Cambridge, MA, USA, 1996).

[16] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*, 10th ed. (Cambridge University Press, Cambridge ; New York, 2010).

[17] K. Thurnhofer-Hemsi, E. López-Rubio, M. A. Molina-Cabello, and K. Najarian, "Radial basis function kernel optimization for Support Vector Machine classifiers," (2020), arXiv:2007.08233 [cs, stat].

[18] M. Schuld and N. Killoran, Physical Review Letters **122**, 040504 (2019), arXiv:1803.07128 [quant-ph].

[19] P. Flach, *Machine learning: The art and science of algorithms that make sense of data*, Machine learning: The art and science of algorithms that make sense of data (Cambridge University Press, New York, NY, US, 2012) pages: xvii, 396.

[20] M. Pal, "Multiclass Approaches for Support Vector Machine Based Land Cover Classification," (2008), arXiv:0802.2411 [cs].

[21] A. Ng, M. Jordan, and Y. Weiss, in *Advances in Neural Information Processing Systems*, Vol. 14 (MIT Press, 2001).

[22] B. Schölkopf, A. Smola, and K.-R. Müller, in *Artificial Neural Networks — ICANN'97*, Lecture Notes in Computer Science, edited by W. Gerstner, A. Germond, M. Hasler, and J.-D. Nicoud (Springer, Berlin, Heidelberg, 1997) pp. 583–588.

[23] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, Journal of Machine Learning Research **12**, 2825 (2011).

[24] P. H., "INTELLIGENT IRRIGATION SYSTEM," (2020).

[25] T. P, "Drug Classification," (2020).

[26] M. A. Little, P. E. McSharry, S. J. Roberts, D. A. Costello, and I. M. Moroz, BioMedical Engineering OnLine **6**, 23 (2007).

[27] K. Pearson, The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science **2**, 559 (1901).

Appendix A: Algorithm

Algorithm 1: Genetic Algorithm for QSVM Kernel Generation

Data: *data*, *maximumGenerations*, *targetFitness*, *numberOfQubits*, *numberOfGenes*, *populationSize*, *optimisationTechnique*
Result: *individual* (*quantumCircuit*)
minimum ← 0;
maximum ← inf;
trainingData ← **trainingDataSplit**(*data*);
validationdata ← **validationDataSplit**(*data*);
testingdata ← **testingDataSplit**(*data*);
while *geneRange* ≥ *numberOfQubits* **do**
  individual ← **Individual**(*numberOfGenes*);
  bestFitness ← 0;
  population ← { } × *populationSize*;
  while *bestFitness* < *targetFitness* & *generations* ≤ *maximumGenetations* **do**
    for *i* ← 0 **to** *populationSize* **do**
      newIndividual ← **mutate**(*individual*, 50%);
      if *optimisationTechnique* == 'Supervised′ **then**
        fitness ← **QSVM**(*newIndividual*, *trainingData*);
      else
        fitness ← **maximumEigenvalue**(*newIndividual*, *trainingData*);
      end
      population[*fitness*] = *newIndividual*;
    end
    bestIndividual ← **maximum**(*population*);
    if *optimisationTechnique* == 'Supervised′ **then**
      fitness ← **QSVM**(*newIndividual*, *trainingData*);
    else
      fitness ← **maximumEigenvalue**(*newIndividual*, *trainingData*);
    end
    if *fitness* ≥ *bestFitness* **then**
      bestFitness ← *fitness*;
      individual ← *bestIndividual*;
      maximum ← *numberOfGenes*;
    else
      minimum ← *numberOfGenes*;
    end
    generations ← *generations* + 1;
  end
  geneRange = *binarySearch*(*min*, *max*);
end
Return *ind*;

Appendix B: Data

Table II. **Dataset Splits**

Dataset	Total Samples	Training Samples	Validation Samples	Testing Samples
Moons	400	192	128	80
XOR	400	192	128	80
Circles	400	192	128	80
Wine	178	85	57	36
Iris	150	72	48	30
Cancer	569	273	182	114
Irrigation	200	96	64	40
Parkinson's	195	93	63	39
Drug Classification	200	96	64	40

Table III. **Moons**

	Supervised GA			**Unsupervised GA**			**PauliZZ**		**RBF**	
Test	**Acc**	**Ent**	**Gates**	**Acc**	**Ent**	**Gates**	**Acc**	**Ent**	**Acc**	**Ent**
1	0.988	0.763	2.0	0.912	0.593	14.0	0.475	0.826	0.675	0.0
2	0.925	0.848	2.0	0.925	0.545	14.0	0.425	0.821	0.675	0.0
3	0.95	0.633	2.0	0.962	0.69	14.0	0.575	0.827	0.7	0.0
4	0.938	0.74	2.0	0.975	0.751	18.0	0.425	0.837	0.625	0.0
5	1.0	0.714	2.0	0.925	0.688	14.0	0.375	0.835	0.8	0.0
6	0.95	0.753	2.0	1.0	0.596	22.0	0.3	0.824	0.65	0.0
7	0.9	0.848	2.0	0.95	0.702	18.0	0.525	0.828	0.75	0.0
8	0.95	0.667	2.0	0.962	0.735	18.0	0.475	0.819	0.775	0.0
9	0.875	0.833	2.0	0.95	0.759	14.0	0.375	0.827	0.8	0.0
10	0.9	0.772	2.0	0.888	0.661	14.0	0.45	0.827	0.7	0.0

Table IV. **XOR**

	Supervised GA			**Unsupervised GA**			**PauliZZ**		**RBF**	
Test	**Acc**	**Ent**	**Gates**	**Acc**	**Ent**	**Gates**	**Acc**	**Ent**	**Acc**	**Ent**
1	0.888	0.766	2.0	0.888	0.698	18.0	0.475	0.826	0.675	0.0
2	0.888	0.752	2.0	0.938	0.776	22.0	0.425	0.821	0.675	0.0
3	0.938	0.767	2.0	0.925	0.726	18.0	0.575	0.827	0.7	0.0
4	0.938	0.834	2.0	0.9	0.765	18.0	0.425	0.837	0.625	0.0
5	0.95	0.846	2.0	0.912	0.766	18.0	0.375	0.835	0.8	0.0
6	0.95	0.837	2.0	0.888	0.714	18.0	0.3	0.824	0.65	0.0
7	0.9	0.572	2.0	0.9	0.735	14.0	0.525	0.828	0.75	0.0
8	0.962	0.483	2.0	0.925	0.665	22.0	0.475	0.819	0.775	0.0
9	0.975	0.462	2.0	0.95	0.575	18.0	0.375	0.827	0.8	0.0
10	0.875	0.847	2.0	0.925	0.736	18.0	0.45	0.827	0.7	0.0

Table V. **Circles**

	Supervised GA			Unsupervised GA			PauliZZ		RBF	
Test	**Acc**	**Ent**	**Gates**	**Acc**	**Ent**	**Gates**	**Acc**	**Ent**	**Acc**	**Ent**
1	1.0	0.863	2.0	0.962	0.506	14.0	0.475	0.826	0.675	0.0
2	0.912	0.863	2.0	0.962	0.746	18.0	0.425	0.821	0.675	0.0
3	0.988	0.863	2.0	0.988	0.735	18.0	0.575	0.827	0.7	0.0
4	0.962	0.865	2.0	0.962	0.745	14.0	0.425	0.837	0.625	0.0
5	0.962	0.863	2.0	1.0	0.648	14.0	0.375	0.835	0.8	0.0
6	0.95	0.865	2.0	0.925	0.749	18.0	0.3	0.824	0.65	0.0
7	0.975	0.864	2.0	0.925	0.672	16.0	0.525	0.828	0.75	0.0
8	0.988	0.865	2.0	0.962	0.496	14.0	0.475	0.819	0.775	0.0
9	0.938	0.862	2.0	0.912	0.548	18.0	0.375	0.827	0.8	0.0
10	0.975	0.863	2.0	0.938	0.697	18.0	0.45	0.827	0.7	0.0

Table VI. **Wine**

	Supervised GA			Unsupervised GA			PauliZZ		RBF	
Test	**Acc**	**Ent**	**Gates**	**Acc**	**Ent**	**Gates**	**Acc**	**Ent**	**Acc**	**Ent**
1	1.0	0.763	2.0	0.778	0.467	46.0	0.475	0.826	0.675	0.0
2	0.972	0.721	2.0	0.889	0.611	62.0	0.425	0.821	0.675	0.0
3	0.972	0.739	2.0	0.833	0.407	30.0	0.575	0.827	0.7	0.0
4	0.917	0.716	2.0	0.972	0.525	30.0	0.425	0.837	0.625	0.0
5	1.0	0.727	2.0	0.833	0.635	62.0	0.375	0.835	0.8	0.0
6	0.972	0.796	2.0	0.889	0.453	50.0	0.3	0.824	0.65	0.0
7	0.972	0.713	2.0	0.75	0.44	54.0	0.525	0.828	0.75	0.0
8	0.944	0.697	2.0	0.889	0.479	32.0	0.475	0.819	0.775	0.0
9	1.0	0.758	2.0	0.889	0.658	30.0	0.375	0.827	0.8	0.0
10	0.972	0.706	2.0	0.944	0.534	30.0	0.45	0.827	0.7	0.0

Table VII. **Iris**

	Supervised GA			Unsupervised GA			PauliZZ		RBF	
Test	**Acc**	**Ent**	**Gates**	**Acc**	**Ent**	**Gates**	**Acc**	**Ent**	**Acc**	**Ent**
1	0.9	0.637	2.0	0.933	0.771	10.0	0.475	0.826	0.675	0.0
2	0.967	0.765	2.0	0.9	0.617	50.0	0.425	0.821	0.675	0.0
3	0.933	0.625	2.0	0.867	0.538	16.0	0.575	0.827	0.7	0.0
4	1.0	0.636	2.0	0.933	0.494	18.0	0.425	0.837	0.625	0.0
5	0.933	0.649	2.0	0.933	0.603	14.0	0.375	0.835	0.8	0.0
6	0.933	0.728	2.0	1.0	0.678	18.0	0.3	0.824	0.65	0.0
7	0.967	0.7	2.0	0.867	0.312	6.0	0.525	0.828	0.75	0.0
8	0.933	0.726	2.0	0.9	0.507	8.0	0.475	0.819	0.775	0.0
9	0.9	0.705	2.0	1.0	0.667	18.0	0.375	0.827	0.8	0.0
10	0.967	0.657	2.0	0.933	0.532	18.0	0.45	0.827	0.7	0.0

Table VIII. **Cancer**

	Supervised GA			**Unsupervised GA**			**PauliZZ**		**RBF**	
Test	**Acc**	**Ent**	**Gates**	**Acc**	**Ent**	**Gates**	**Acc**	**Ent**	**Acc**	**Ent**
1	0.895	0.657	2.0	0.842	0.672	14.0	0.475	0.826	0.675	0.0
2	0.886	0.518	10.0	0.868	0.51	14.0	0.425	0.821	0.675	0.0
3	0.895	0.532	6.0	0.86	0.601	32.0	0.575	0.827	0.7	0.0
4	0.886	0.7	2.0	0.877	0.411	26.0	0.425	0.837	0.625	0.0
5	0.877	0.671	2.0	0.851	0.538	34.0	0.375	0.835	0.8	0.0
6	0.904	0.438	6.0	0.877	0.412	42.0	0.3	0.824	0.65	0.0
7	0.93	0.628	6.0	0.816	0.582	62.0	0.525	0.828	0.75	0.0
8	0.912	0.576	2.0	0.86	0.615	50.0	0.475	0.819	0.775	0.0
9	0.912	0.652	6.0	0.886	0.455	94.0	0.375	0.827	0.8	0.0
10	0.86	0.627	2.0	0.877	0.591	94.0	0.45	0.827	0.7	0.0

Table IX. **Irrigation**

	Supervised GA			**Unsupervised GA**			**PauliZZ**		**RBF**	
Test	**Acc**	**Ent**	**Gates**	**Acc**	**Ent**	**Gates**	**Acc**	**Ent**	**Acc**	**Ent**
1	1.0	0.841	2.0	1.0	0.598	6.0	0.475	0.826	0.675	0.0
2	0.95	0.776	2.0	0.975	0.773	4.0	0.425	0.821	0.675	0.0
3	1.0	0.786	2.0	0.975	0.795	6.0	0.575	0.827	0.7	0.0
4	1.0	0.775	2.0	1.0	0.691	2.0	0.425	0.837	0.625	0.0
5	0.95	0.768	2.0	1.0	0.77	6.0	0.375	0.835	0.8	0.0
6	0.925	0.777	2.0	1.0	0.787	6.0	0.3	0.824	0.65	0.0
7	0.95	0.779	2.0	0.9	0.794	6.0	0.525	0.828	0.75	0.0
8	0.975	0.795	2.0	1.0	0.702	2.0	0.475	0.819	0.775	0.0
9	0.975	0.793	2.0	0.975	0.794	4.0	0.375	0.827	0.8	0.0
10	0.95	0.834	2.0	0.975	0.773	4.0	0.45	0.827	0.7	0.0

Table X. **Parkinson's**

	Supervised GA			**Unsupervised GA**			**PauliZZ**		**RBF**	
Test	**Acc**	**Ent**	**Gates**	**Acc**	**Ent**	**Gates**	**Acc**	**Ent**	**Acc**	**Ent**
1	0.872	0.823	14.0	0.744	0.666	30.0	0.475	0.826	0.675	0.0
2	0.692	0.493	18.0	0.641	0.424	126.0	0.425	0.821	0.675	0.0
3	0.795	0.65	26.0	0.872	0.697	30.0	0.575	0.827	0.7	0.0
4	0.872	0.487	62.0	0.846	0.578	62.0	0.425	0.837	0.625	0.0
5	0.795	0.575	14.0	0.846	0.607	30.0	0.375	0.835	0.8	0.0
6	0.872	0.646	22.0	0.795	0.79	54.0	0.3	0.824	0.65	0.0
7	0.821	0.439	62.0	0.667	0.439	126.0	0.525	0.828	0.75	0.0
8	0.821	0.392	62.0	0.821	0.503	128.0	0.475	0.819	0.775	0.0
9	0.744	0.458	126.0	0.769	0.691	54.0	0.375	0.827	0.8	0.0
10	0.872	0.546	30.0	0.897	0.674	54.0	0.45	0.827	0.7	0.0

Table XI. **Drug Classification**

	Supervised GA			**Unsupervised GA**			**PauliZZ**		**RBF**	
Test	**Acc**	**Ent**	**Gates**	**Acc**	**Ent**	**Gates**	**Acc**	**Ent**	**Acc**	**Ent**
1	0.9	0.762	14.0	0.9	0.841	22.0	0.475	0.826	0.675	0.0
2	0.85	1.106	10.0	0.875	0.523	86.0	0.425	0.821	0.675	0.0
3	0.925	0.88	6.0	0.875	0.718	42.0	0.575	0.827	0.7	0.0
4	0.85	0.896	14.0	0.95	0.866	62.0	0.425	0.837	0.625	0.0
5	0.775	0.937	14.0	0.9	0.792	26.0	0.375	0.835	0.8	0.0
6	0.925	0.695	10.0	0.775	0.492	126.0	0.3	0.824	0.65	0.0
7	0.925	0.865	6.0	0.875	1.033	30.0	0.525	0.828	0.75	0.0
8	0.875	0.715	26.0	0.9	0.686	62.0	0.475	0.819	0.775	0.0
9	0.875	1.097	12.0	0.825	1.01	30.0	0.375	0.827	0.8	0.0
10	0.825	0.703	6.0	0.9	0.535	30.0	0.45	0.827	0.7	0.0