Quantum Information Processing (2023) 22:89
https://doi.org/10.1007/s11128-023-03837-1

Data rotation and its influence on quantum encoding

Daniel Sierra-Sosa[1] · Soham Pal[2] · Michael Telahun[3]

Received: 28 November 2021 / Accepted: 10 January 2023 / Published online: 23 January 2023

Abstract
Parametric quantum machine learning (QML) has been vastly studied over the last several years. These algorithms rely on hybrid implementations, where quantum methods define the models, and the parameters are update on classical devices. The encoding of classical data into quantum states within the Hilbert space is fundamental to training these hybrid models; this can be achieved in a number of ways. In this work, we focus on two of these methods, amplitude encoding and encoding via a second-order Pauli feature map. We compared their performances across two near-term QML models, quantum support vector classifier and variational quantum classifier. We found that amplitude encoding is significantly resilient to classical transformations of data. This work additionally introduces the concept of a rotation, applied to classical data as a preprocessing step. In our results, we observe that other encoding methods can significantly benefit from certain Cartesian rotations of the data. We expand this rotation to a larger $n - D$ dataset and show the method's performance.

Keywords Quantum computing · Machine learning · Encoding · Preprocessing · Distribution · Rotation

1 Introduction

In a broad sense, quantum machine learning (QML) is the overlap of quantum information science and machine learning [1]. The term QML can mean a number of things: developing quantum-inspired machine learning algorithms, using quantum computers

Daniel Sierra-Sosa, Soham Pal and Michael Telahun have contributed equally to this work.

✉ Daniel Sierra-Sosa
sierra-sosa@hood.edu

1 Department of Computer Science and Information Technology, Hood College, Frederick, MD, USA

2 Department of Physics and Astronomy, Iowa State University, Ames, IA, USA

3 Department of Computer Science and Engineering, University of Louisville, Louisville, KY, USA

to "speed-up" certain machine learning tasks, or using classical machine learning to find patterns in quantum processes, depending on the practitioner [2]. Here we use QML to mean the use of a quantum computer, as a part of a larger hybrid quantum-classical system, to analyze classical or quantum data and infer patterns from such data. It still remains to be seen if quantum computers will demonstrate any advantage over classical computers for machine learning tasks [3, 4]. However, recent algorithmic developments, the availability of noisy intermediate scale quantum (NISQ) devices, and open-source quantum computing software have shown some paths forward [5].

It will undoubtedly take some time for quantum computing (QC) and QML to naturally progress from the current NISQ era to one that is fault-tolerant. This evolution will be guided by exploiting the capabilities of NISQ devices and understanding what sort of problems near-term QML algorithms are suitable for. During this time of NISQ, it is important to benchmark near-term QML algorithms and devise ways to improve their performance. Therefore, in this work we explore the influence of (1) statistical structures of classical data, (2) classical transformations of data, and (3) quantum encoding methods on the performance of near-term QML algorithms. We want to emphasize that this is not an analysis of model performance by fine-tuning model hyperparameters. Rather, this is an analysis of the influence of data on near-term QML algorithms.

For this purpose, we choose two quantum encoding methods—amplitude encoding [6] and encoding via a second-order Pauli feature map [7], there are number of variants for embedding of data into quantum states. This work does not aim to add a new embedding to the solutions present but primarily compare and show the resilience to different embedding solutions. We do our experiments using two near-term QML algorithms, the quantum support vector classifier (QSVC), and variational quantum classifier (VQC), over five different distributions, changing their separation scale and rotating the data. We perform our experiments on quantum simulators from the Qiskit and PennnyLane services [8, 9].

The remainder of this work is as follows: we discuss the models in Sect. 2, the binary classification datasets in Sect. 3, and the details of the training method and the results in Sect. 4. And finally we conclude our findings in Sect. 5.

2 Methods

The primary components of this work come from training and evaluating different statistical distributions of datasets after applying: (1) rotation transformation going from 0 to 2π radians in steps of $\pi/12$ radians and (2) different encoding methods or feature maps to the classical datasets. To show that our results are not specific to a model, we train and evaluate two different QML algorithms—QSVC and VQC—on each dataset, applying these different rotations and feature maps.

We have implemented a bottom-up workflow that follows from a standard classical approach for data preparation to quantum modeling. It includes the steps necessary for the quantum methods to be applied before measurements and the computation of the cost function of the model. This workflow diagramed in Fig. 1 shows the difference between the QSVC and VQC encoding methods. For both methods, the data is first

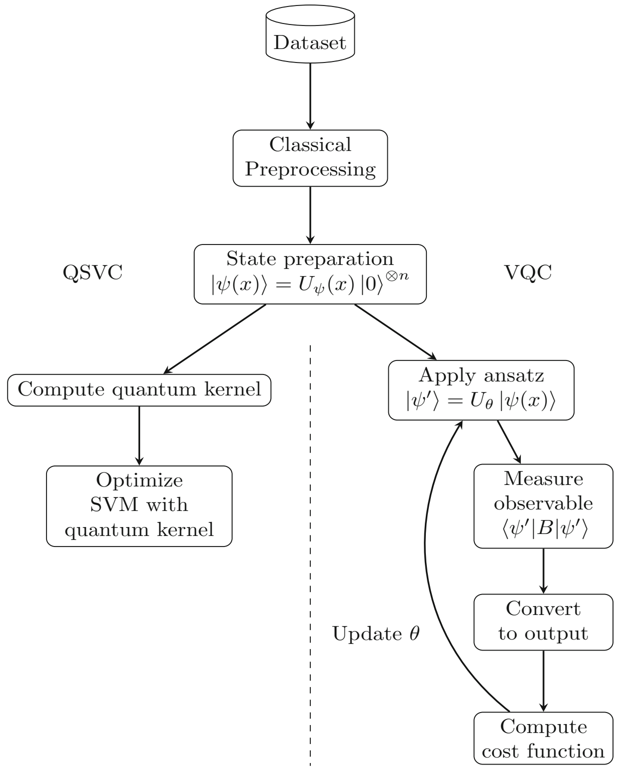

Dataset
Classical Preprocessing
State preparation $|\psi(x)\rangle = U_\psi(x)\,|0\rangle^{\otimes n}$
QSVC
VQC
Compute quantum kernel
Optimize SVM with quantum kernel
Apply ansatz $|\psi'\rangle = U_\theta\,|\psi(x)\rangle$
Measure observable $\langle\psi'|B|\psi'\rangle$
Convert to output
Compute cost function
Update θ

Fig. 1 The general workflow of our implemented methodology. We use a similar bottom-up approach for both QSVC and VQC to achieve the analysis presented. Both algorithms require the classical data to be encoded into quantum states via quantum feature maps. The two techniques differ following the various steps for state preparation primarily due to their intrinsic design

loaded. Data are then preprocessed, in the work below we will outline that in this step we manipulate scale, rotation, and distribution shift. This step is not restricted to our methods and generally could include other steps such as binning. Before the quantum processing, we perform state preparation which encodes the classical data into quantum states which represent the data in a way that can be ingested by the models. We use two embedding methods, amplitude encoding and a Pauli feature map, both of which are expanded upon in Sect. 2.2.

For the QSVC branch of the workflow, prior to training, a quantum kernel must be computed in order to determine the optimal decision boundary much like the classical SVM. This kernel ultimately must be executed via the quantum computer, we provide and explain a SVM and QSVC algorithm below in Sect. 2.1.1. Once the quantum kernel is created, the optimization or training follows.

In the case of VQC, the branch begins with the application of an ansatz to the prepared state. The ansatz is a single unitary that is a tunable parameter within the model. A measurement is then performed, collapsing the quantum state, using the Kronecker product for the expected value. The measure is then converted to be used in the cost function of the optimization. The cost function is then computed; these steps are expended upon in Sect. 2.1.2. We then return to the application of the ansatz and repeat until the number of optimization iterations is exhausted.

2.1 Classifiers

Both QSVC and VQC are hybrid techniques that have been applied in several different applications since they have come around as successful applications of QML on NISQ devices. As the name suggests, QSVC is a quantum extension of the celebrated classical machine learning algorithm, support vector machine (SVM), and VQC belongs to the group of variational quantum algorithms which includes variational quantum eigensolvers (VQE) and variational quantum linear solver (VQLS). The VQC is a supervised method that is circuit centric meaning the model learns the parameters passed to the gates applied to the model's circuit. Both of these algorithms are parametric, that is they can output models optimized to a given dataset. We briefly describe both these models in the following subsection.

2.1.1 Quantum support vector classifier (QSVC)

The SVM algorithm is as follows [10]:

1. Map the input data to a high-dimensional feature space via some nonlinear mapping, chosen *a priori*.
2. Construct an optimal hyperplane in the high-dimensional feature space that separates the classes.

In practice, instead of using a nonlinear mapping, SVM maps the input data into a high-dimensional, possibly infinite, feature space by the so-called kernel trick. Formally it can be understood as follows [10–12]. Given N training samples $(x_1, y_2), \ldots, (x_n, y_n)$ with $x_i \in \mathbb{R}^p$ and $y_i \in \{-1, +1\}$ the SVM finds $\beta \in \mathbb{R}^p$ and $\beta_0 \in \mathbb{R}$ such that $y_i = \text{sign}[x_i^T \beta + \beta_0]$ is true for most data samples. There can be multiple such pairs (β, β_0), each defining a hyperplane separating the two classes. The optimal hyperplane is the one that maximizes the margin between the training points of the two classes. Thus, the SVM solves the following primal optimization problem:

$$\begin{gathered}\min_{\beta,\beta_0} \frac{1}{2}\beta^T \beta + C e^T \xi \\ \text{subject to } \xi_i \geq 0,\ y_i(x_i^T \beta + \beta_0) \geq 1 - \xi_i,\ i = 1, \ldots, N,\end{gathered} \tag{1}$$

where e is a vector of all ones, and C is the regularization factor that penalizes for samples that are on the wrong side of the margin. The dual to this is:

$$\begin{gathered}\max_{\alpha} e^T \alpha - \frac{1}{2}\alpha^T Q \alpha \\ \text{subject to } y^T \alpha = 0,\ 0 < \alpha_i < C,\ i = 1, \ldots, N,\end{gathered} \tag{2}$$

where α_i are the dual coefficients, Q is an $N \times N$ positive semidefinite matrix defined $Q_{ij} = y_i y_j k(x_i, x_j)$ with kernel $k(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ and the map $\phi : \mathbb{R}^p \to \mathcal{S}$, $\dim(\mathcal{S}) \geq p$. The beauty of this approach is that we need not be concerned with f at all and we just need to specify a suitable kernel. The dual can be solved easily in comparison with the primal. The solution function, also known as decision boundary, for a given sample x is

$$f(x) = \sum_{i}^{N} \alpha_i y_i k(x, x_i) + \beta_0, \tag{3}$$

with α_i and β_0 determined by solving $y_i f(x_i) = 1$ subject to the constraints in Eq. (2). The class of x is determined by sign$[f(x)]$.

QSVC thus follows naturally from SVM but with a quantum kernel $k(x_i, x_j) = \left|\langle\psi(x_i)\rangle\psi(x_j)\right|^2$, i.e., the kernel is executed on a quantum computer [7, 13]. Here the input vector x_i has been encoded into a n-qubit quantum state $|\psi(x_i)\rangle$ by some unitary transformation $U_\psi(x_i)$ on the $|0\rangle^{\otimes n}$ state. Once the kernel matrix has been computed, the QSVC can be trained similarly as the classical SVM. The solution function for a given sample x is same as in Eq. (3), with $k(x, x_i) = |\langle\psi(x)\rangle\psi(x_i)|^2$. When using some kernels, classical support vector machines are invariant to transformations such as translation, rotation or scaling [14, 15], the quantum counterpart on the other hand is not invariant, therefore susceptible to optimization.

2.1.2 Variational quantum classifier

Variational quantum algorithms, including VQC, are based on training a parameterized unitary (ansatz) in a hybrid quantum classical by minimizing a suitable cost function [16]. The cost function (and its gradients, if required) is estimated on a quantum computer, and a classical optimizer is used to train the parameters of the quantum circuit. For quantum advantage, the cost function should be difficult to compute using a classical computer; this has yet to be shown in any recent work.

Given N training samples $(x_1, y_2), \ldots, (x_n, y_n)$ with $x_i \in \mathbb{R}^p$ and $y_i \in \{-1, +1\}$, the VQC algorithm is as follows [13, 17]:

1. Encode the input vector x_i into a n-qubit state quantum state by applying a unitary transformation $U_\psi(x_i)$ to the $|0\rangle^{\otimes n}$ state: $|\psi(x_i)\rangle = U_\psi(x_i)|0\rangle^{\otimes n}$.
2. Apply an ansatz U_θ to the input state: $|\psi'\rangle = U_\theta|\psi(x_i)\rangle$. θ are then the trainable parameters.
3. Measure the expectation value of a chosen observable B:

$$\mathbb{E}(B) = \langle U_\theta \psi(x_i)|B|U_\theta \psi(x_i)\rangle, \tag{4}$$

   and convert the measurement to an output $\hat{y}_\theta(x_i) = \text{sign}[\mathbb{E}(B)]$. Usually B is chosen to be the Kronecker product of Pauli operators.

4. Compute the cost function:

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N} L_s(y_i, \hat{y}_\theta(x_i)), \tag{5}$$

   where $L_s(y_i, \hat{y}_\theta(x_i))$ is the cost per sample. The cost per sample function $L_s(\cdot, \cdot)$ is chosen *a priori*.
5. Minimize the cost function by iteratively tuning the parameters θ using a classical optimizer.

The class of any new data sample x is determined by sign$[\langle U_{\theta'}\psi(x)|B|U_{\theta'}\psi(x)\rangle]$, where $\theta^{'} = \arg\min_\theta L(\theta)$.

2.2 State preparation and feature maps

The first step in both QSVC and VQC is encoding classical data into quantum states . There is a computational cost to encode the information from classical devices to quantum devices, and on many occasions, this impairs the algorithm efficiency. In general, a quantum algorithm will be considered efficient if it uses a polynomial number of qubits and gates, and classical data can be encoded with different algorithms that could be efficient in terms of the employed number of qubits or in terms of the amplitudes of the quantum states [13], some efforts to speed up the quantum state preparation have been made and reported [18, 19]; nonetheless, the discussion about the classical or quantum computational complexity is out of the scope of this work. We will include in our analysis and describe two widely use encoding techniques: amplitude encoding and second-order Pauli feature map. The encoding also known as state preparation is done using a quantum feature map $\psi : \mathbb{R}^p \rightarrow \mathbb{C}^{2n}$, where n is the number of qubits. The quantum feature map transforms $x \in \mathbb{R}^p$ to a quantum state in $|\psi(x)\rangle$ in the complex Hilbert space $\mathbb{C}^{2n}$ via some parametric unitary transformation $U_\psi(x)$, i.e., $|\psi(x)\rangle = U_\psi(x)|0\rangle$. Albeit different encoding schemes exist [20, 21], in this work we consider two such quantum feature maps: amplitude encoding and second-order Pauli feature map.

2.2.1 Amplitude encoding

In amplitude encoding a p-dimensional datapoint x is translated to the amplitudes of an n-qubit quantum state $|\psi_x\rangle$ [6]:

$$x \mapsto |\psi(x)\rangle = \sum_{i=1}^{2^n} x_i|i\rangle, \tag{6}$$

where $|i\rangle$ are the n-qubit computational basis states, and x_i are the amplitudes. This process is accounted for via two classical preprocessing steps:

1. If $p < 2^n$, then the input vector x is padded with zero features such that the dimension of x is 2^n.

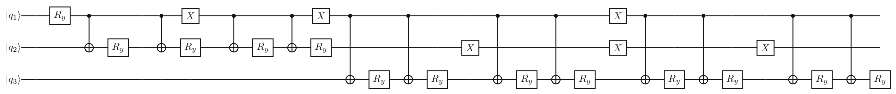


Fig. 2 A single implementation of amplitude encoding for a three qubit circuit. This circuit provides the means for a dataset with up to eight classical features. State preparation via amplitude encoding requires the application of Eq. 6 to the classical data before applying this circuit. This three-qubit amplitude encoding circuit then encodes the given arbitrary state $|\psi\rangle$ given in Eq. 7

2. The input vector x is normalized.

The input vector is then transformed to $|\psi(x)\rangle$ by a sequence of uniformly controlled rotations [22]. To implement amplitude encoding, the rotation on a given qubit q_s is controlled by all possible states of the previous qubits $q_1, \ldots, q_{s-1}$ by using multiple controlled rotations, the rotation angle is defined by the association of the vector v^i representing the ith classical sample with a choice of a rotation angle β_i related to the amplitudes of the original state as described by [6]. The state $|\psi\rangle$ is then defined as:

$$|\psi\rangle = R(v^i, \beta)|q_1...q_{s-1}\rangle|q_s\rangle \tag{7}$$

The state $|\psi\rangle$ is prepared as a circuit "cascading" n R_y rotations, where n represents the power in binary for encoding the vector v^i. In Fig. 2, we present a three-qubit amplitude encoding circuit example where up to 8-dimensional classical data can be encoded into quantum states.

2.2.2 Encoding via second-order Pauli feature map

The general Pauli feature map is a data encoding circuit that consists of repetitions of the unitary $\mathcal{U}_\Phi(x) = U_{\Phi(x)} H^{\otimes n} U_{\Phi(x)} H^{\otimes n}$ [7], where H is the Hadamard gate and

$$U_{\Phi(x)} = \exp\Big(i \sum_{S\subseteq[n]} \phi_S(x) \prod_{i\in S} P_i\Big), \tag{8}$$

where $P_i \in \{I, X, Y, Z\}^{\otimes k}$ for $k = 1, \ldots, n$, and the index S, standing for subsets of $[n]$ with size $\leq k$, describes the connectivity between different qubits or datapoints. The coefficients ϕ_S are defined by a nonlinear function to fully separate the feature map on the different number of classes j while allowing the function $U_{\Phi(x)}$ to be implemented efficiently

$$\phi_S = \begin{cases} x_0 \text{ if } k = 1, \\ \prod_{j\in S}(\pi - x_j) \text{ otherwise }, \end{cases} \tag{9}$$

where x_i is the i-th entry in the input vector x. In this work, we focus and show results for a specific Pauli feature map obtained by setting $k = 2$, $P_{\{i\}} = Z$, and $P_{\{i,j\}} = ZZ$. This is implemented as `ZZFeatureMap` in Qiskit, and as `IQPEmbedding` in PennyLane. In the following, whenever "Pauli feature map" is mentioned we refer to this specific feature map, and not the general one. The authors chose this feature map instead of others, such as the first-order `ZFeatureMap` due to its similar overall behavior to encoding the datasets.

3 Dataset

In this work, we are less interested in analyzing the performance of the models by fine-tuning the model hyperparameters but instead build our analysis on the robustness of the chosen encoding method. This includes the rotations to the datasets and their effects. To perform our analysis, we prepared five datasets in total each of which are synthetic and described in detail within Sect. 3.1. In each model, we keep the hyperparameters fixed throughout all our experiments and evaluate their performance across the datasets.

3.1 Distributions

For our analysis, we generate two-dimensional binary classification datasets by sampling from different statistical distributions. These datasets are manually created using a generalization of the `make_blobs` generator method from `scikit-learn` [23] and are reproducible per distribution given. For this purpose, we have chosen five statistical distributions, each of which is defined by a location parameter μ, and a scale parameter $b > 0$ (or a shape parameter $\kappa > 0$). The location parameter determines the shift of the distribution, and the scale (shape) parameter determines the spread (shape) of the distribution. With $z = x - \mu$, the probability density functions of these distributions are defined as follows:

1. Gumbel: $p(x) = \frac{1}{b} e^{-(z+e^{-z})}$,
2. Laplace: $p(x) = \frac{1}{2b} e^{-|z|/b}$,
3. Logistic: $p(x) = \frac{1}{b} \frac{s}{(1+s)^2}$, where $s = e^{-z/b}$,
4. Normal: $p(x) = \frac{1}{\sqrt{\pi} b'} e^{-(z/b')^2}$, where $b' = \sqrt{2} b$,
5. von Mises: $p(x) = \frac{1}{2\pi I_0(\kappa)} e^{\kappa \cos z}$, where $I_0(\kappa)$ is the modified Bessel function of order zero.

For each of these datasets, we have 60 datapoints clustered into two classes. The location of each cluster is chosen by a uniform random number generator from the range $[-7.5, 7.5]$. The scale (shape) parameter for each cluster is fixed to 1.0. Figure 3 shows the raw datasets sampled from these distributions. In the case of higher dimensions, we generated a 4-dimensional dataset with normal distribution and 300 datapoints, again clustered into two classes, but in the range $[-1.0, 1.0]$.

3.2 Scales

Since the normal distribution frequently appears in scientific applications, it behooves us to give special attention to it. For that purpose, we generate a second set of binary

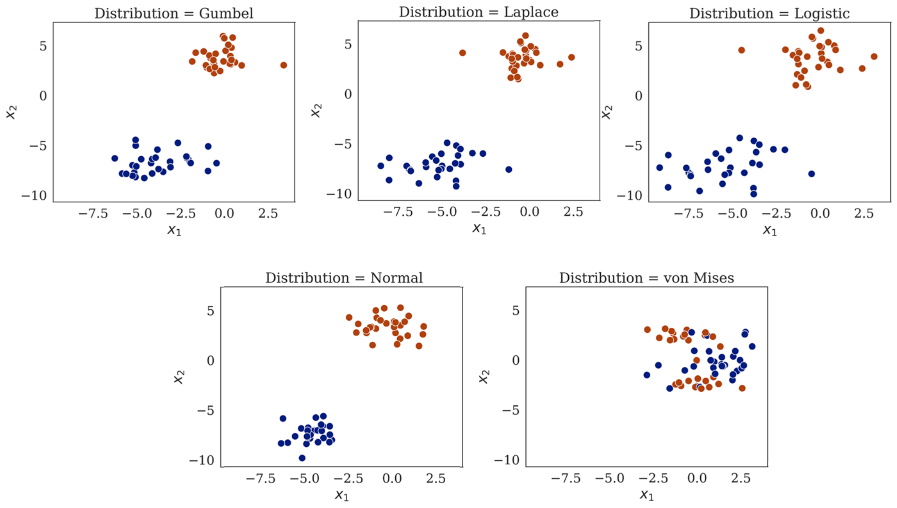


Fig. 3 Synthetic datasets created with the generalized `make_blobs` generator by sampling from five different statistical distributions: Gumbel, Laplace, Logistic, Normal and von Mises. Each dataset consists of 60 points clustered into two classes. All except the dataset sampled from the von Mises distribution are linearly separable

classification datasets by changing the scale parameter of the normal distribution. In addition to the dataset described above, we generate five additional datasets by sampling from normal distributions with $b \in [0.5, 3.5, 5.5, 7.5, 9.5]$. As above each of these datasets have 60 datapoints clustered into two classes, and the location of each cluster is chosen by a uniform random number generator from the range $[-7.5, 7.5]$. Figure 4 shows the raw datasets.

3.3 Rotations

For each of the raw two-dimensional datasets, we create rotated versions by transforming the input vector x as:

$$x \mapsto x' = R(\theta)(x - x_0) + x_0, \tag{10}$$

where $R(\theta)$ is the rotation matrix,

$$R(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}, \tag{11}$$

and x_0 is the origin. In all our experiments, we set $x_0 = (0, 0)$, and vary the angle θ from 0 radians to 2π radians in steps of $\pi/12$ radians. Thus, for each raw two-dimensional dataset we have 24 rotated datasets. This idea is easily extended to $D > 2$-dimensional datasets by fixing $D - 2$ dimensions and rotating the remaining two dimensions. For the n-dimensional rotations, we followed the work presented by Aguilera and Perez-Aguila in [24]. A set of vertices is defined as:

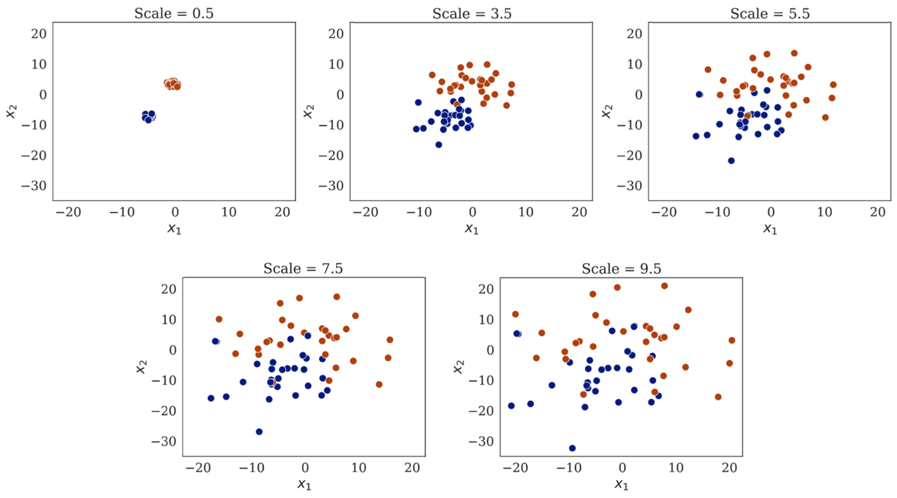
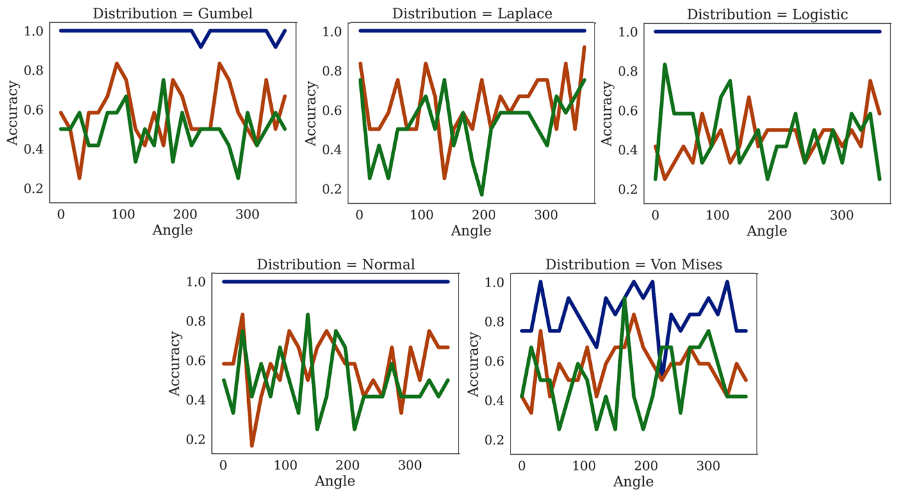


Fig. 4 Synthetic datasets created with `make_blobs` by changing the scale parameter of the normal distribution. For the normal distribution, the scale parameter coincides with the standard deviation. Each dataset consists of 60 points clustered into two classes. As the standard deviation increases the datasets become less and less linearly separable

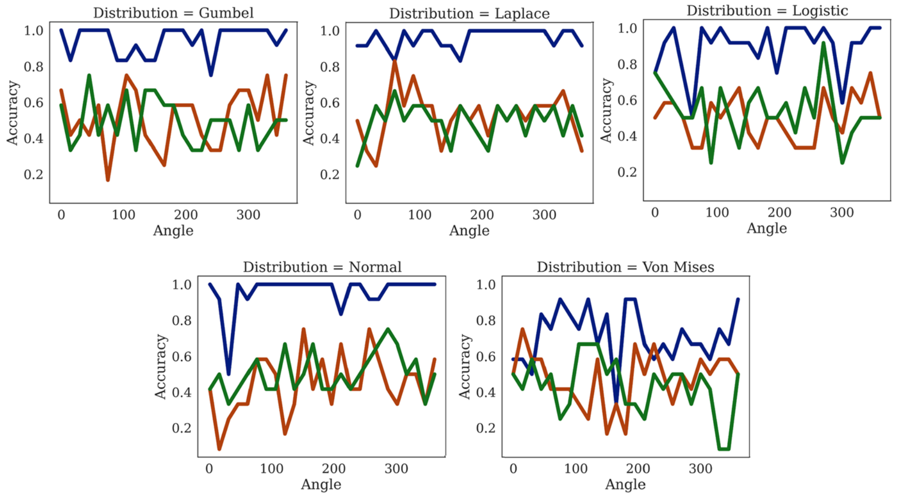
$$v_k = \begin{pmatrix} v_{k(1,1)} & v_{k(1,2)} & \cdots & v_{k(1,n)} \\ v_{k(2,1)} & v_{k(2,2)} & \cdots & v_{k(2,n)} \\ \vdots & \vdots & \ddots & \vdots \\ v_{k(n-1,1)} & v_{k(n-1,2)} & \cdots & v_{k(n-1,n)} \end{pmatrix}, \tag{12}$$

where v_0 is the set of original vertices before transformation, is rotated following the transformation defined by $v_k = v_{k-1} \cdot M_k$, being M_k a transformation matrix. A rotation provided a given angle is conducted around a $n-2$-dimensional subspace represented by a simplex T with vertices v_0. Then, the first transformation matrix is computed as $M_1 = T(-a)$ where a is the first row of v_0. Then in the k^{th} step, the element $v_{k_{(r,c)}}$ for a given row r and column c can vanish using $M_k = R_{c,c-1}(arctan(v_{k-1_{(r,c)}}, v_{k-1_{(r,c-1)}}))$ where $R_{a,b}$ is the rotation matrix defined as

$$R_{a,b}(\theta) = r_{i,j} \begin{cases} r_{a,a} = \cos\theta \\ r_{b,b} = \cos\theta \\ r_{a,b} = -\sin\theta \\ r_{b,a} = \sin\theta \\ r_{j,j} = 1, j \neq a, j \neq b \\ r_{i,j} = 0, \text{elsewhere} \end{cases} \tag{13}$$



(a) QSVC

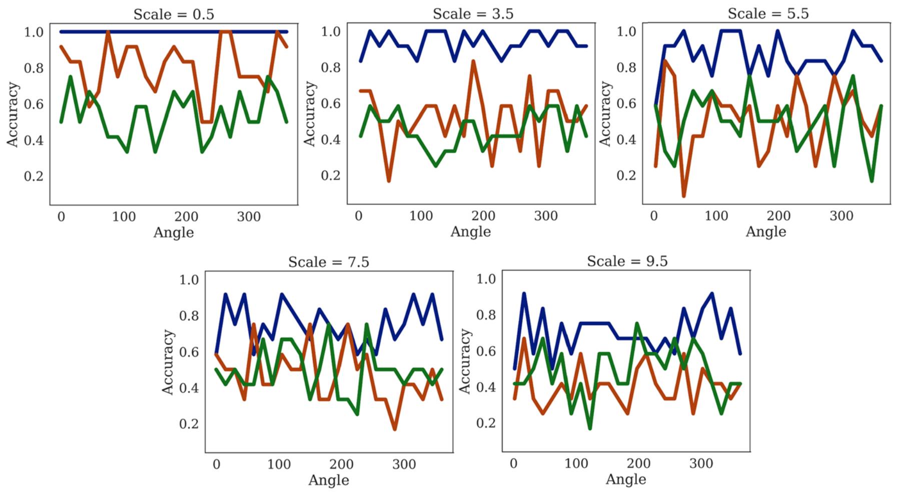


(b) VQC

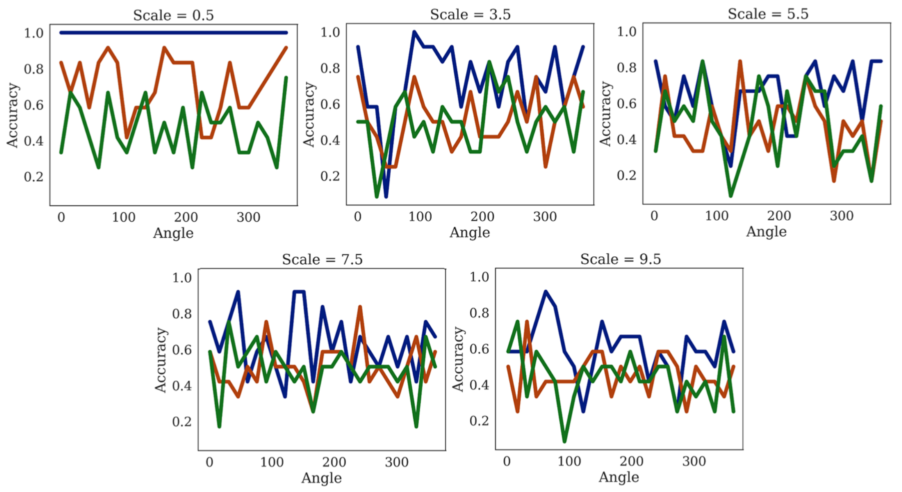
Fig. 5 Accuracy of the QML algorithms with the distributions dataset. The colors blue, red, and green are, respectively, amplitude encoding, `IQPEmbedding`, and `ZZFeatureMap`. Each boxplot is constructed from 25 datapoints, the raw dataset and its 24 rotations

4 Results

As mentioned above in all our experiments, we rotate the raw datasets in steps of $\pi/12$ radians from 0 to 2π radians. We encode each of those rotated datasets once by amplitude encoding, once by `IQPEmbedding`, and once by `ZZFeatureMap`. We randomly split each rotated dataset in a 4:1 ratio into a training set and a test set. We train the QML models on the training sets and test their performances using the



(a) QSVC



(b) VQC

Fig. 6 Accuracy of the QML algorithms with the scales dataset. The colors blue, red, and green are, respectively, amplitude encoding, `IQPEmbedding`, and `ZZFeatureMap`. Each boxplot is constructed from 25 datapoints, the raw dataset and its 24 rotations

test sets. Each model is trained with each of the encoded training sets. Thus, for each rotated dataset there exist three QSVC models and three VQC models (one for each encoding). We then compute the average accuracy of each model with respect to the corresponding test set.

Figures 5 and 6 show the accuracy summaries for the distributions and scales datasets, respectively. We see that amplitude encoding is more resilient to the rotations than either of the implementations of Pauli feature map encoding. In all cases considered, when the data are encoded by Pauli feature map, the accuracy of the QML

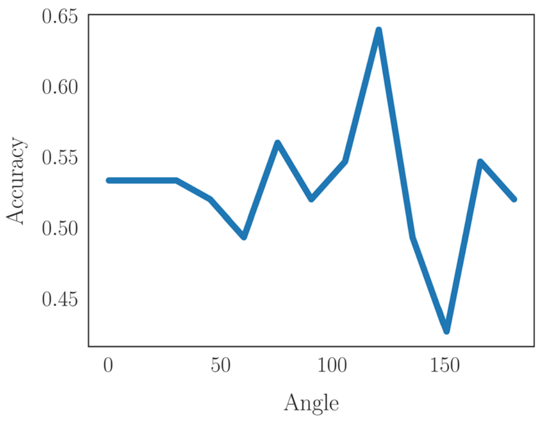


Fig. 7 Accuracy of the VQC algorithm with 4-dimensional Normal distributions dataset

algorithms is significantly boosted by certain rotations of the data. On the other hand, for amplitude encoding the effects of rotation are most prominent when the datasets are mostly not linearly separable. Based on the results obtained in the two-dimensional experiments, we selected the parameters and model to assess the dataset rotation influence on QML performance. The accuracies obtained when training and testing a VQC algorithm with a 4-dimensional normal distribution also rotated in $\pi/12$ radians are summarized in Fig. 7.

The exact angles of rotation at which the accuracy peaks are unimportant for this analysis, since that angle varies between distributions and implementations, and undoubtedly would not hold for natural data. Rather we want to emphasize that we can utilize a rotation of the dataset as a preprocessing step. By rotating the dataset, there is potential to improve the classification results without any prior knowledge in the domains of quantum computing or machine learning. This should not be confused with feature engineering new information but should be treated as an alternate representation of the data which improves classification results.

5 Conclusion

Comparing the two encoding methods, it appears that in general the QML algorithms perform more consistently and with less deviation when the data are encoded with amplitude encoding. The lack of variance between data points is particularly noticeable when observing the results of the QSVC models. It is also clear that although the encoding allows for increased performance of the classifier, there are still several rotations that created more harm than good. These attribute for the large whiskers on the box plots, specifically when looking at the von Mises dataset. This behavior is not unlike other preprocessing methods where a chosen normalization or discretization can have adverse effects on the capability of a classifier to poorly optimize a given dataset.

Albeit these results are sub-par when compared with those that can be obtained by conventional machine learning techniques, the presented results based on rotations are apparent and provide an applicable enhancement. Granted we did not choose to concern ourselves with an exact rotation degree, an optimization to this method could be created to reduce the search space for the optimal range of rotation. In the future, we aim to analyze additional changes to data that increase performance of QML techniques and measure their capacity as quantum preprocessing methods.

Data Availability All the data used in this paper were generated using Python scikit-learn library; no other data sources were employed to obtain the results presented.

Declaration

Conflict of interest All authors declare that they have no conflicts of interest to disclose.

References

1. Schuld, M., Petruccione, F.: Machine Learning with Quantum Computers. Quantum Science and Technology, Springer, New York (2021)
2. Dunjko, V., Wittek, P.: A non-review of quantum machine learning: trends and explorations. Quantum Views **4**, 32 (2020)
3. Maheshwari, D., Garcia-Zapirain, B., Sierra-Sosa, D.: Quantum machine learning applications in the biomedical domain: a systematic review. IEEE Access **10**, 80463–80484 (2022)
4. Zidan, M., Abdel-Aty, A., Younes, A., Zanaty, E., El-Khayat, I., Abdel-Aty, M.: A novel algorithm based on entanglement measurement for improving speed of quantum algorithms. Appl. Math. Inf. Sci **12**(1), 265–269 (2018)
5. Biamonte, J., Wittek, P., Pancotti, N., Rebentrost, P., Wiebe, N., Lloyd, S.: Quantum machine learning. Nature **549**(7671), 195–202 (2017)
6. Schuld, M., Bocharov, A., Svore, K.M., Wiebe, N.: Circuit-centric quantum classifiers. Phys. Rev. A **101**, 032308 (2020)
7. Havlíček, V., Córcoles, A.D., Temme, K., Harrow, A.W., Kandala, A., Chow, J.M., Gambetta, J.M.: Supervised learning with quantum-enhanced feature spaces. Nature **567**, 209–212 (2019)
8. Abraham, H., Akhalwaya, I.Y., Aleksandrowicz, G., Alexander, T., Alexandrowics, G., Arbel, E., Asfaw, A., Azaustre, C., Barkoutsos, P., Barron, G., Bello, L., Ben-Haim, Y., Bishop, L.S., Bosch, S., Bucher, D., CZ, Cabrera, F., Calpin, P., Capelluto, L., Carballo, J., Chen, C.-F., Chen, A., Chen, R., Chow, J.M., Claus, C., Cross, A.W., Cross, A.J., Cruz-Benito, J., Cryoris, Culver, C., Córcoles-Gonzales, A.D., Dague, S., Dartiailh, M., Davila, A.R., Ding, D., Dumitrescu, E., Dumon, K., Duran, I., Eendebak, P., Egger, D., Everitt, M., Fernández, P.M., Frisch, A., Fuhrer, A., Gacon, J., Gadi, Gago, B.G., Gambetta, J.M., Garcia, L., Garion, S., Gawel-Kus, Gil, L., Gomez-Mosquera, J., de la Puente González, S., Greenberg, D., Gunnels, J.A., Haide, I., Hamamura, I., Havlicek, V., Hellmers, J., Herok, Ł., Horii, H., Howington, C., Hu, W., Hu, S., Imai, H., Imamichi, T., Iten, R., Itoko, T., Javadi-Abhari, A., Jessica, Johns, K., Kanazawa, N., Karazeev, A., Kassebaum, P., Krishnan, V., Krsulich, K., Kus, G., LaRose, R., Lambert, R., Latone, J., Lawrence, S., Liu, P., Mac, P.B.Z., Maeng, Y., Malyshev, A., Marecek, J., Marques, M., Mathews, D., Matsuo, A., McClure, D.T., McGarry, C., McKay, D., Meesala, S., Mezzacapo, A., Midha, R., Minev, Z., Morales, R., Murali, P., Müggenburg, J., Nadlinger, D., Nannicini, G., Nation, P., Naveh, Y., Nick-Singstock, Niroula, P., Norlen, H., O'Riordan, L.J., Ollitrault, P., Oud, S., Padilha, D., Paik, H., Perriello, S., Phan, A., Pistoia, M., Pozas-iKerstjens, A., Prutyanov, V., Pérez, J., Quintiii, Raymond, R., Redondo, R.M.-C., Reuter, M., Rodríguez, D.M., Ryu, M., Sandberg, M., Sathaye, N., Schmitt, B., Schnabel, C., Scholten, T.L., Schoute, E., Sertage, I.F., Shi, Y., Silva, A., Siraichi, Y., Sivarajah, S., Smolin, J.A., Soeken, M., Steenken, D., Stypulkoski, M., Takahashi, H., Taylor, C., Taylour, P., Thomas, S., Tillet, M., Tod, M., de la Torre, E., Trabing, K., Treinish, M., TrishaPe, Turner, W., Vaknin, Y., Valcarce, C.R., Varchon, F., Vogt-Lee, D., Vuillot, C., Weaver, J., Wieczorek, R., Wildstrom, J.A., Wille, R., Winston, E., Woehr, J.J., Woerner, S., Woo, R., Wood, C.J., Wood, R., Wood, S., Wootton, J., Yeralin, D., Yu, J., Zdanski, L., Zoufalc, anedumla,

azulehner, bcamorrison, drholmie, fanizzamarco, kanejess, klinvill, merav-aharoni, ordmoj, tigerjack, yang.luh, yotamvakninibm: Qiskit: An Open-source Framework for Quantum Computing (2019)
9. Bergholm, V., Izaac, J., Schuld, M., Gogolin, C., Alam, M.S., Ahmed, S., Arrazola, J.M., Blank, C., Delgado, A., Jahangiri, S., et al.: Pennylane: Automatic differentiation of hybrid quantum-classical computations. arXiv:1811.04968 (2018)
10. Vapnik, V.N.: The Nature of Statistical Learning Theory. Springer, New York (2013)
11. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer Series in Statistics, 2nd edn. Springer, New York (2009)
12. Bishop, C.M.: Pattern Recognition and Machine Learning, Information Science and Statistics. Springer, New York (2016)
13. Schuld, M., Fingerhuth, M., Petruccione, F.: Implementing a distance-based classifier with a quantum interference circuit. EPL (Europhys. Lett.) **119**(6), 60002 (2017)
14. Abe, S.: On invariance of support vector machines. In: Proceedings of the 4th International Conference on Intelligent Data Engineering and Automated Learning (2003)
15. Haasdonk, B., Burkhardt, H.: Invariant kernel functions for pattern analysis and machine learning. Mach. Learn. **68**(1), 35–61 (2007)
16. Cerezo, M., Arrasmith, A., Babbush, R., Benjamin, S.C., Endo, S., Fujii, K., McClean, J.R., Mitarai, K., Yuan, X., Cincio, L., Coles, P.J.: Variational Quantum Algorithms (2020)
17. Mitarai, K., Negoro, M., Kitagawa, M., Fujii, K.: Quantum circuit learning. Phys. Rev. A **98**, 032309 (2018)
18. Möttönen, M., Vartiainen, J.J., Bergholm, V., Salomaa, M.M.: Transformation of quantum states using uniformly controlled rotations. quant-ph/0407010 (2004)
19. Araujo, I.F., Park, D.K., Petruccione, F., da Silva, A.J.: A divide-and-conquer algorithm for quantum state preparation. Sci. Rep. **11**(1), 1–12 (2021)
20. Schuld, M., Sweke, R., Meyer, J.J.: Effect of data encoding on the expressive power of variational quantum-machine-learning models. Phys. Rev. A **103**(3), 032430 (2021)
21. Adhikary, S., Dangwal, S., Bhowmik, D.: Supervised learning with a quantum classifier using multi-level systems. Quantum Inf. Process. **19**(3), 1–12 (2020)
22. Möttönen, M., Vartiainen, J.J., Bergholm, V., Salomaa, M.M.: Quantum circuits for general multiqubit gates. Phys. Rev. Lett. **93**, 130502 (2004)
23. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: machine learning in python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
24. Aguilera, A., Pérez-Aguila, R.: General n-dimensional rotations. In: WSCG (2004)