Good quantum kernels and where to find them

A. J. Ferreira–Martins[*] and C. Almeida[†]
UFABC

(Dated: October 24, 2021)

The framework is essentially made up of 5 components:

I. PRE-PROCESSING

We built this framework as a generic tool, to be applied to arbitrary datasets. However, given the nature of the operations performed, which includes a large number of simulations/executions of quantum circuits, it is very important that dimensionality of the data that we work with is somehow adjusted. Otherwise, simulation or execution on NISQ-era hardware becomes impractical.

Thus, in order to guarantee generality, we included to the framework a procedure of sub-sampling, with the goal of providing the user with ways to reduce the dimensionality of the data prior to the actual quantum operations are performed – that is, as a pre-processing step.

When it comes to the construction of quantum kernels, it is very important to reduce as much as possible both the number of observations (training/testing examples) as well as the number of features/predictors. Indeed, the number of observations dictate how many quantum circuits must be executed in order to construct the kernel matrix. As simulating/executing these circuits may be costly, the smaller the number of observations, the better, as this implies a smaller number of circuit simulations/executions. On the other hand, the number of features determines the number of qubits making up each circuit. As the number of qubits increase, it becomes more costly or even impossible to simulate/execute the quantum circuits. Therefore, the smaller the number of features, the better, as this implies in a smaller number of qubits per circuit. This is why reductions in both dimensionalities is desirable.

Now, as it's always the case when such dimensionality reduction techniques are employed, it is important to do so whilst retaining most of the original information. With this in mind, we propose two options for this procedure:

- **Subsampling from Determinantal Point Processes (DPPs)**: with DPPs we are able to construct subsamples of the original data, by selecting rows and/or columns from the data matrix in a repulsive manner, which means that it favors the sampling of rows/columns which are different among themselves, thus producing a representative, diverse subsample. We use the dppy python library. Please refer to its documentation, or other relevant literature like this one for further details;
- **Using Principal Components Analysis (PCA)**: with PCA, we are able to compute the so-called Principal Components, which are linear combinations of the original data matrix columns with a very particular property: they represent the directions which explain the data variance the most. Therefore, by choosing just a few of the first principal components, we hope to reduce the dimensionality of the features space quite dramatically, whilst retaining at least part of the original information, as all original features are present in the PC's linear combination. We use sklearn's implementation. Notice that, in contrast to DPPs, we use PCA only as a way to reduce the number of features, not observations.

These pre-processing routines are presented as options for the user, so they're not mandatory. This means that, if one desires, it is in principle possible to run the framework for an arbitrary dataset with an arbitrary number of features and observations — all tools (except from some visualization tools) were made sufficiently generic as to accommodate any dimensionality. In the next section, we elaborate on the implications of the dimensionality of the data for the actual search procedures.

II. GRID/RANDOM SEARCH

The framework is meant to be a tool to systematically test different feature map architectures leading to quantum kernel matrices associated to a given dataset. To implement such a systematic approach, we provide 2 options: a grid search as well as a random search approach.

In the grid search approach, we consider and explicitly test all elements in the full space of parameters combinations. Conversely, in the random search approach we consider only a fixed number of combinations of the full space, thus explicitly using a much smaller number of options to test. Grid search does access the full space of combinations, and because of that, it implies in much higher computational costs. Random search, on the other hand, is expected to yield good results whilst reducing dramatically the computational cost of the search.

If the grid search approach is chosen, we strongly advise against using a number of features larger than 4 and a number of observations larger than 100, as this would make the execution times of the framework way too long. Now, if one desires to use dimensionalities larger than the

[*] andre.jfmdm@gmail.com
[†] caio.almeida.ads@gmail.com

ones we advised for grid search, we strongly suggest the use of random search.

Now, in order to discuss the actual space of parameters to be tested, it is important to discuss how we shall construct the quantum feature maps.

III. QUANTUM FEATURE MAPS

For the contraction of quantum feature maps leading to quantum kernels, we shall use the PauliFeatureMap class from qiskit, and we'll vary the following parameters:

- 'eps': the number of repeated blocks of the chosen feature map architecture;
- 'entanglement': specifies the entanglement structure associated to the feature map architecture;
- 'paulis': a list of strings for pauli operators to be used in the feature map.

The general map to be implemented is:

$$U_{\Phi(\vec{x})} = \exp\left(i \sum_{S \subseteq [n]} \phi_S(\vec{x}) \prod_{i \in S} P_i \right) \ , \tag{1}$$

where

$$\phi_S(\vec{x}) = \begin{cases} x_0 \text{ if } k = 1 \\ \prod_{j \in S} (\pi - x_j) \text{ otherwise} \end{cases} \tag{2}$$

And a particular feature map architecture is determined by a particular combination of the aforementioned parameters.

In order to exhaustively search for feature maps leading to good quantum kernels, we construct the full space of parameters allowed for `paulis` and `entanglement_list`. For `reps`, one could have, in principle, an infinite countable space (`reps` $\in \mathbb{N}$). However, we restrict this space to reps $\in \{1, 2, 3\}$, given that repetitions larger than 4 become increasingly difficult to and/or execute on NISQ-era hardware.

Once the space of parameters for each one of the three aforementioned parameters is defined, we construct all possible combinations of the 3 parameters values. However, some of these combinations, despite different, lead to identical feature maps. It's very important that such redundancies are eliminated, in order to reduce the space of parameters combinations as much as possible (after all, we perform an exhaustive search, so that if we can reduce the space of possibilities, we definitely should do so, for better efficiency). Please refer to this notebook, in which we perform experiments and tests in order to argue how and in which cases it is possible to remove such redundancies.

The functions which produce the space of parameters combinations (which we refer to as "parameters grid"), as well as all important functions, are in the `qsvm_utils.py`, described in great detail.

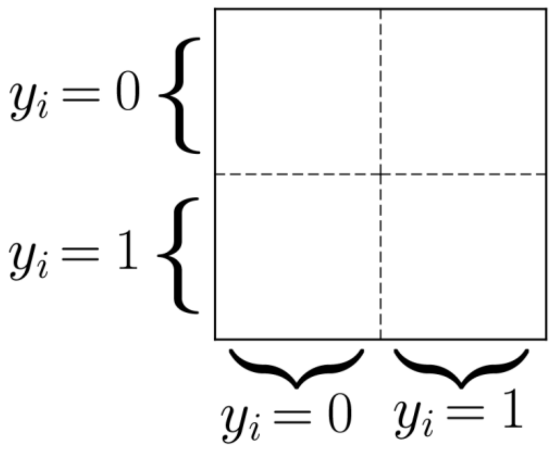
Up to now, we described how we will systematically construct quantum kernels. But, the framework is meant to actually determine, from all options considered in the search, which one yields the best result. In order to do so, we must have a way to actually evaluate the quality of a given quantum kernel.

IV. EVALUATION METRIC

In this project, we are interested in *binary classification* performed by classical SVM classifiers, using a quantum pre-computed kernel matrix. Given that, it is very reasonable that a kernel is a good one if it captures the following similarity structure: observations which belong to the same class are *similar* among themselves; and observations which are from different classes are **dissimilar** among themselves. In mathematical terms, consider two observations x_i and x_j. If they are from the same class ($y_i = y_j = 0$ or $y_i = y_j = 1$), we'd expect the respective Gram/kernel matrix component to be close to 1 $G_{ij} = \kappa_\Phi(x_i, x_j) \approx 1$; and if they are from different classes ($y_i \neq y_j$), we'd expect $G_{ij} = \kappa_\Phi(x_i, x_j) \approx 0$ (in this project, we consider a normalization such that $0 < G_{ij} < j \forall i, j$).

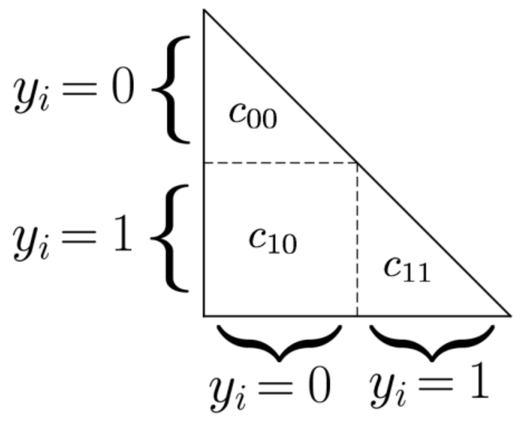
In other words, a kernel will be good if it can *separate* observations properly with respect to their class — after all, that's the goal of classification!

Building on that, we propose an intuitive and visual way to assess the quality of kernel. First, order the observations so that observations of the same class are grouped together. This structures the Gram matrix in blocks:

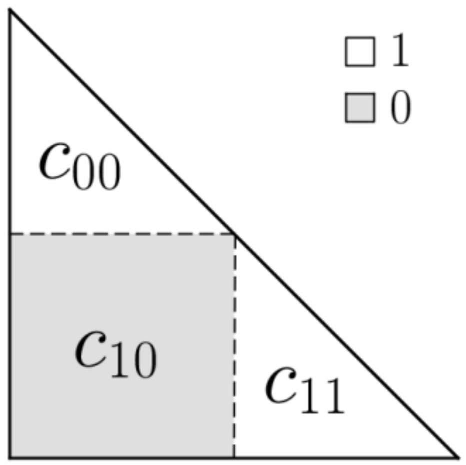


Given that the training Gram matrix is symmetric ($G_{ij} = G_{ji}$), we can focus only on its lower diagonal:

Now, a "perfect" kernel matrix would have all of its components either 0 (in the bottom-left square region



containing points of different classes) or 1 (in the remaining two triangles which contain points of the same class), that is,



Real kernel matrices won't be that perfect, but, if a kernel matrix is a good one, we at least expect to be able to identify such a structure when it is plotted (like a heatmap, for instance). Such a visualization perfectly illustrates if the kernel is able to separate well the different classes or not. A "good" kernel matrix would look like this:

fig4.

On the other hand, for a "bad" kernel matrix, it wouldn't be possible to identify the structure. It would look like this:

fig5.

Thus, by merely looking at plotted kernel matrices, we would be able to tell good ones from bad ones. That's a rather simple, intuitive and well-justified visual criterion. However, having to visually inspect possibly hundreds or thousands of plots really doesn't seem like a great idea. Thinking of that, we proposed a quantitative metric which aims at capturing precisely this same visual structure, but in a single number. Referring to the regions c_{00}, c_{11} and c_{01} as shown in Fig. IV, the proposed metric is:

$$\Delta_\mu = \frac{1}{2}\left(\mu_{c_{00}} + \mu_{c_{11}}\right) - \mu_{c_{01}} \ , \tag{3}$$

where μ is simply the average of the elements in the respective region. We call each average a "density", which is why we refer to the metric Δ_μ as "densities difference".

Notice that for a perfect kernel (illustrated in Fig. IV), we'd have $\Delta_\mu = 1$, which is therefore the maximum value for Δ_μ. A quite bad kernel would be the opposite of the one in Fig. IV, such that $\mu_{c_{00}} = \mu_{c_{11}} = 0$ and $\mu_{c_{10}} = 1$ (that is, it considers observations of the same class quite dissimilar, whilst observations of different class are quite similar). In this case, we'd have $\Delta_\mu = -1$. Given those extreme scenarios, it is clear that:

$$-1 \leq \Delta_\mu \leq 1 \ . \tag{4}$$

This gives us a very forward way to valuate a kernel (and, most importantly, compare different kernels): the closer Δ_μ is to 1, the better the kernel; the closer Δ_μ is to -1, the worst. With this, we have established a meaningful metric to evaluate the kernels, and all we have to do is to calculate it for every quantum kernel matrix that is calculated in the grid/random search process, and then sort the kernels according to Δ_μ to pick the best ones!

Now, a very reasonable question is whether or not this proposed metric indeed is meaningful. That is, does it really captures the aforementioned visual behavior? Further, does it really capture the quality of a kernel itself?

In order to answer this question, we proposed a further step on the framework: we not only compute several kernels and their respective Δ_μ, but we also feed the resulting quantum kernel matrix as a pre-computed kernel to a classical SVM classifier estimator (we use the implementation from sklearn, train and evaluate it on test data, which yields several classification metrics (namely, precision, recall and f1 score). With these classification metrics, we are then able to answer the following question: is Δ_μ correlated with the classification metrics? Indeed, it is reasonable that we expect that a "good kernel" must correlate strongly (and positively) with a good model (which is, after all, the final goal). Thus, if such a strong positive correlation is observed between Δ_μ and the classification metrics, we could safely say that indeed it does capture the quality of a kernel.

As can be seen in the results notebooks, we found quite strong positive correlations between Δ_μ (`density_diff`) and the classification metrics. Therefore, we conclude that Δ_μ is indeed a good metric for the quality of a kernel.

Once the grid/random search executes through all the combinations, it produces a dataframe with all the aforementioned metrics (which is saved for later reference), as well as the respective feature map parameters. With this data, one is able to analyze several aspects of the results. We include such analysis as the final component of the framework.

V. ANALYTICS AND VISUALIZATION

As a final step, we produce some important analysis of the produced results, which serve as a way to check for the proposed procedure as well as provide some insight into what makes a good kernel actually good; secondly, we also provide some interesting visualization tools.

First, we calculate and plot the correlation matrix of the results. In it, we have Δ_μ as well as the classification metrics and two other quantitative metrics which were proposed to quantify the feature map parameter "paulis": the first one is "pauli quantity", which quantifies how many pauli gates were considered in the parameter value; the second one is "pauli diversity", which quantifies how many different paulis where considered. For example, for the parameter value ["X", "ZX"], we'd have pauli quantity equals 3, and "pauli diversity" equals 2. From our results, we found some really interesting points:

- Indeed, Δ_μ is a good kernel quality metric, as explained above; - Δ_μ is almost uncorrelated to "reps", the number of block repetitions on the feature map; - Δ_μ is strongly and negatively correlated with "pauli quantity", indicating that fewer paulis lead to better kernels (that's quite interesting!); - Δ_μ also has a negative correlation with "pauli diversity", althoug not too strong. Also interesting!

Each result above is very interesting and could lead to lengthy discussions, which we won't do here. However, we hope this shows how much our framework does lead to some really interesting questions and possible insights on what makes good kernels good — and that's exactly one of our main goals with the project!

Finally, we also provide two interesting visualization tools: first, a way to visualize the decision boundary generated by the classical SVM trained with the quantum pre-computed kernel. This is very interesting because it allows one to study how the linear margin classifier on feature space projects onto the input space. Unfortunately, of course, this can only be done for 2-dimensional input spaces. The second visualization tool uses Qiskit's tool to visualize what happens with a given observation from the input space as it is mapped to Hilbert space by feature maps. The idea here is to visualize "the action" of the feature map (here we do what was schematically represented in the cover picture of the project). Although multi-qubit visualization is not the easiest task, he hope that this tool helps one understand what exactly a feature map does.

And that's all! This is the framework we propose. We hope all points were made clear (remember, the full code is thoroughly commented for clarity of all major points!), and that it really is something useful for the QML research community. There are natural next steps for this project, and they will be addressed soon. In the mean time, if there's any question or suggestions, we're happy to receive them!