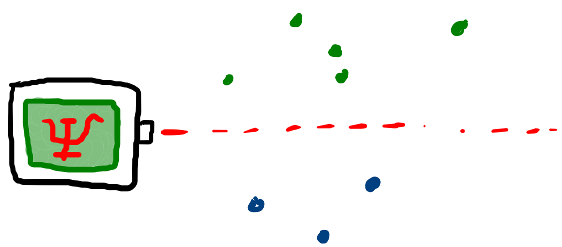
QUANTUM MACHINE LEARNING & KERNEL METHODS



Based on Schuld, 2101.11020

David Wakeham
UBC Strings Talk

Sep 10, 2021

WHAT IS QML?

- Quantum machine learning (QML) is ill-defined.
- It could mean learning quantum data, building models using quantum algorithms, or both!

ALGORITHM / MODEL

		classical	quantum
DATA	classical	CC	CQ
	quantum	QC	QQ

- The top right is the hype corner. I'll focus on concrete example of supervised learning.

I

SOME CLASSICAL BACKGROUND

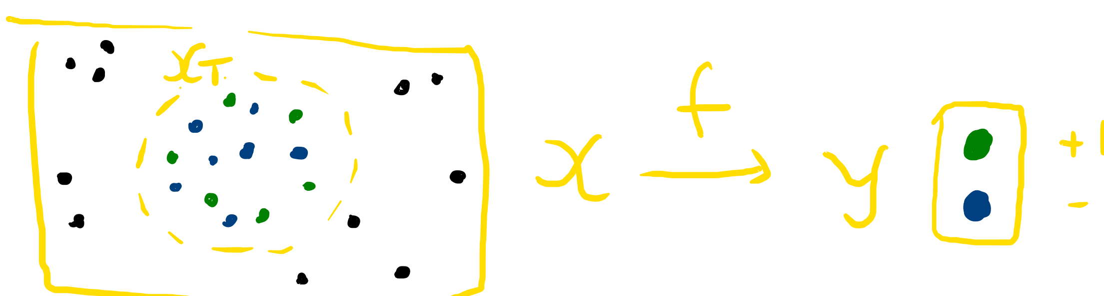
SUPERVISED LEARNING

- Basic idea: data points $x \in \mathcal{X}$ we are trying to classify, e.g. pictures of donuts vs. bagels.

donut
-1

bagel
$+1$

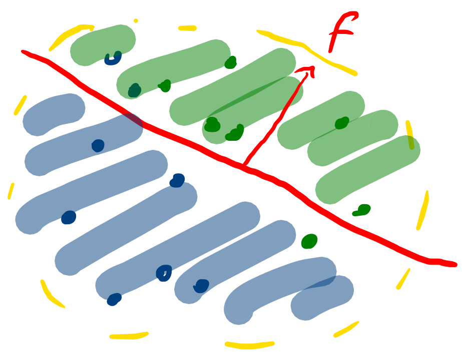
- In supervised learning, we are given access to a training set $x^m \in \mathcal{X}_T$ of M labelled data points.



- We use this training data to create a model $f(x)$ which guesses the label $y \in \mathcal{Y}$ for all of $\mathcal{X}$.

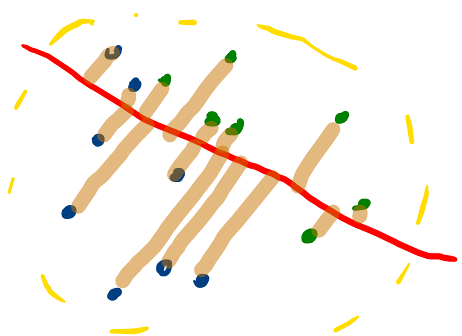
LINEAR MODELS

- Lots of ways to build this predictor $f: \mathcal{X} \to \mathcal{Y}$, but simplest class is set of linear models.
- Typically data space $\mathcal{X}$ is a vector space, and I can separate training data using a hyperplane:



hyperplane
$f = \text{const.}$

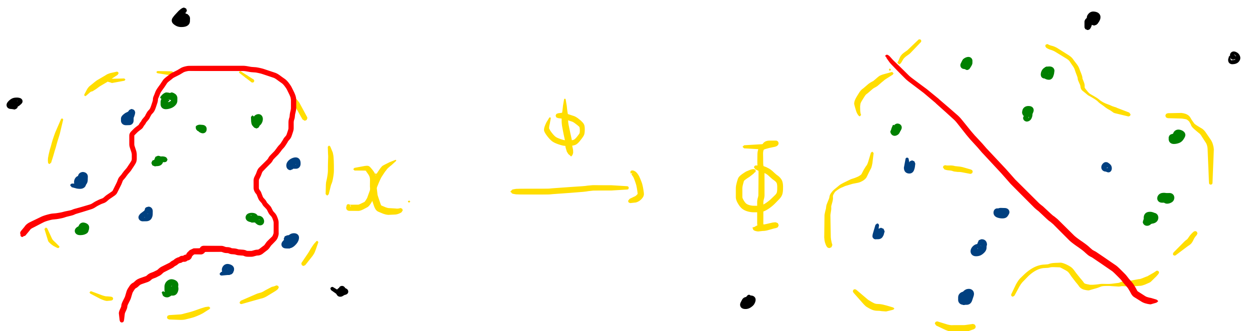
- The hyperplane minimizes an empirical risk function $\hat{R}(f)$:



minimize $\hat{R}(f)$

NONLINEAR MODELS

- Donuts and bagels aren't well-linearly separated in image space.

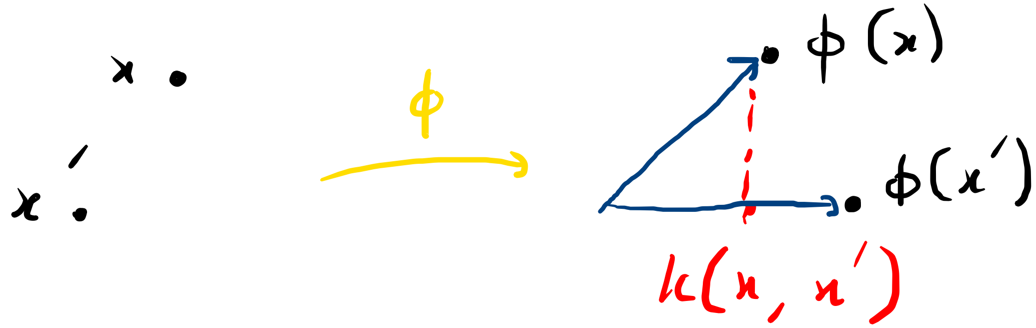


nonlinear linear

- Workaround: map data to "feature space" $\phi(x) \in \Phi$, where a linear decision boundary is better.
- Unfortunately, useful feature spaces are high-dimensional! Explicitly mapping/manipulating feature vectors is costly.

THE KERNEL TRICK

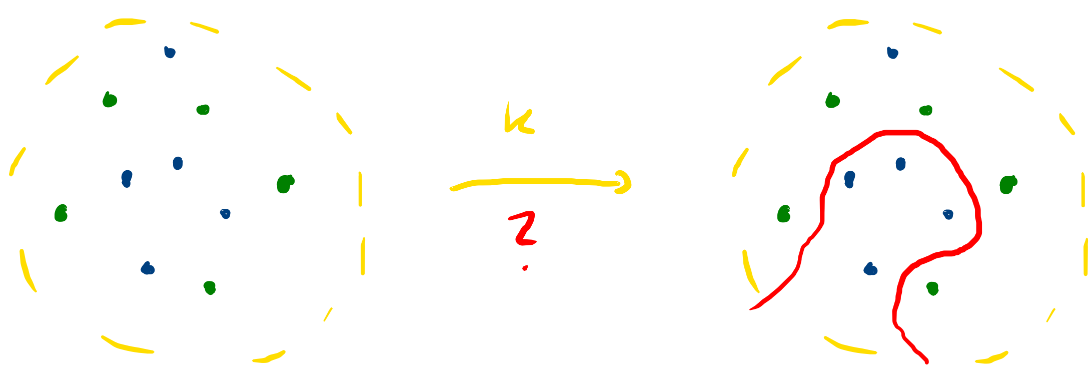
- But there's a hack for avoiding costly explicit manipulation. We work with inner products instead of feature vectors:



- The function taking a pair of data point and giving the inner product of feature vectors is the kernel:

$$k(x, x') = \langle \phi(x), \phi(x') \rangle_{\Phi}.$$

- These are just numbers! But what can we do with them?



REPRODUCING KERNELS

- It's useful to talk about hyperplanes themselves. Mathematically, we use the reproducing kernel Hilbert space (RKHS).
- Use the kernel to map each data point x to a function λ_x:

$$x \longmapsto \lambda_x(\cdot) = k(x, \cdot).$$

$$\begin{tikzcd} \mathcal{X} \arrow[d, "\lambda"] \\ \hom(\mathcal{X}, \mathbb{R}) \end{tikzcd}$$

- Next, define an inner product which reproduces the kernel:

$$\langle \lambda_x, \lambda_{x'} \rangle_F = k(x, x').$$

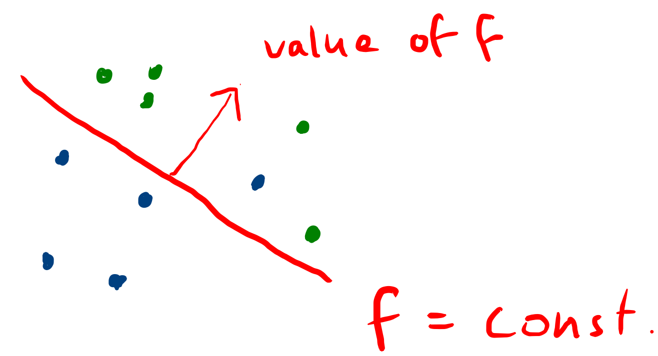
$$\begin{tikzcd} \mathcal{X}^2 \arrow[d, "(\lambda,\lambda)"'] \arrow[dr, "k"] & \\ \hom^2 \arrow[r, "\langle \cdot,\cdot \rangle_F"'] & \mathbb{R} \end{tikzcd}$$

- To create a Hilbert space F, we take the linear span of λ_x, make $\langle \cdot, \cdot \rangle_F$ bilinear, and complete limits as usual.

THE REPRESENTER THEOREM

- The RKHS is (in essence) the space of linear functionals on the span of $\phi(x)$, i.e. the space of linear models:

$$f = \sum_x \alpha^x \lambda_x \longleftrightarrow$$



- We might expect that the models involving only training data $x^m \in \mathcal{X}_T$ are optimal in some sense. This is true!

- More precisely, the Representer Theorem states that any empirical risk-minimizing $f^* \in F$ takes the form

$$f^*_m(x) = \sum_m \alpha^m \lambda_{x^m}(x) = \sum \alpha^m K(x^m, x).$$

They are expansions in training data!

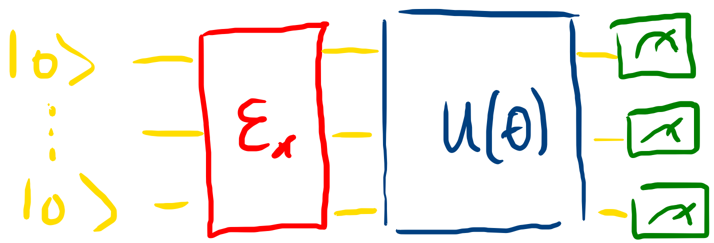
II

QUANTUM MODELS

QUANTUM CIRCUITS

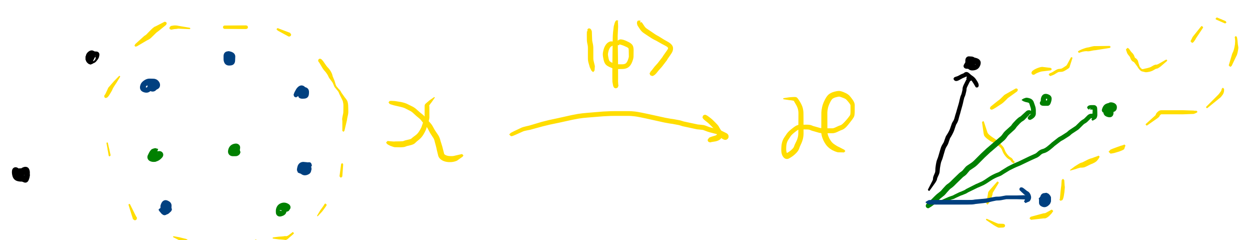
- Let's forget about kernels and linear models, and go quantum. Our goal is to train a circuit to do our classification.

- Scheme: apply an encoding unitary $\mathcal{E}_x$, then some unitary $U(\theta)$ we have trained with $\mathcal{X}_T$, and finally measure.



- The Hilbert space $\mathcal{H}$ of the circuit acts like a feature space:

$$x \longmapsto |\phi(x)\rangle = \mathcal{E}_x |0\rangle .$$



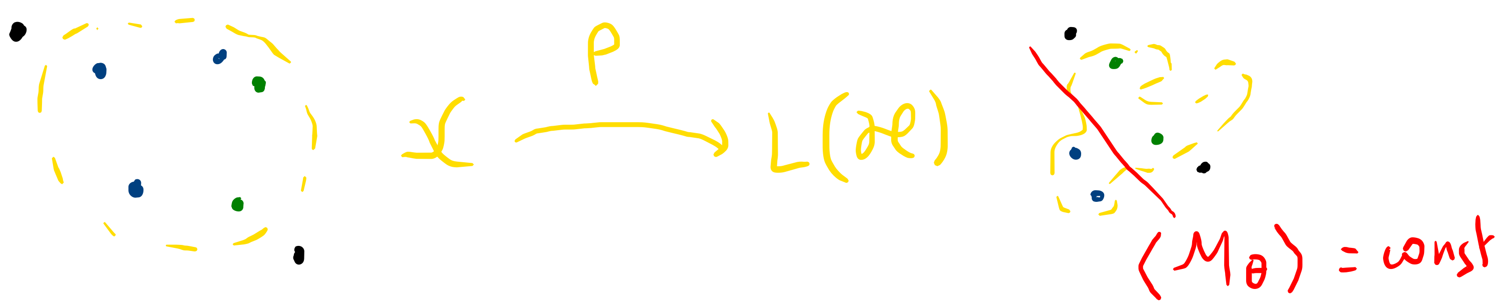
MODELS AND MEASUREMENTS

- What do models look like in the "feature space" $\mathcal{H}$? Let's massage it until it looks linear:

$$
\begin{aligned}
f(x) &= \langle \phi(x) | U(\theta)^\dagger U(\theta) | \phi(x) \rangle \\
&= \mathrm{tr}\left[|\phi(x)\rangle\langle\phi(x)| \, U(\theta)^\dagger U(\theta) \right] \\
&= \mathrm{tr}\left[\rho_x \mathcal{M}_\theta \right] \\
&= \langle \rho_x , \mathcal{M}_\theta \rangle_{HS} .
\end{aligned}
$$

- This is a linear model, but not in $\mathcal{H}$! Rather, on the space of linear operators $L(\mathcal{H})$. This suggests a feature map

$$
x \mapsto \rho_x = |\phi(x)\rangle\langle\phi(x)| .
$$



THE QUANTUM KERNEL

- Kernels come from feature maps. A data-dependent circuit gives a feature map. What is the "quantum" kernel?

$$\kappa(x, x') = \langle \rho_x , \rho_{x'} \rangle_{HS}$$
$$= \mathrm{tr}[\rho_x \rho_{x'}]$$
$$= |\langle \phi(x) | \phi(x') \rangle|^2,$$

i.e. the overlap squared for pure state encoding.

- More generally, the RKHS $F_{\mathcal{H}}$ is the (completed) span of

$$\lambda_x = \kappa(x, \cdot)$$
$$= \mathrm{tr}[\rho_x \, \cdot]$$
$$= \langle \cdot \rangle_x .$$

In other words, $F_{\mathcal{H}}$ consists of expectations in "data states".

ENCODING AND KERNELS

- Since $\kappa(x,x') = \mathrm{tr}[\rho_x \rho_{x'}]$, we need to specify the encoding $x \mapsto \rho_x$ to get an actual kernel. The choice depends on $\mathcal{X}$.
- Some examples:

data	encoding	kernel	cost
$x \in \{0,1\}^n$	$\lvert x \rangle\langle x \rvert$	$\delta_{x,x'}$	$\mathcal{O}(n)$
$x = \sum_i x_i \lvert i \rangle \in \mathbb{C}^{2^n}$	$x x^\dagger$	$\lvert x^\dagger x' \rvert^2$	$\mathcal{O}(n)$
$x = (x_i) \in [0, 2\pi]^n$	$\bigotimes_i e^{i \frac{x_i}{2} \underset{\text{Pauli}}{Y}} \lvert 0 \rangle$	$\prod_i \lvert \cos(x_i - x_i') \rvert^2$	$\mathcal{O}(n)$
$x = (x_i) \in \mathbb{R}^n$	$\bigotimes_i e^{x_i (a^\dagger - a)} \lvert 0 \rangle$ (Fock)	$e^{-\lvert x - x' \rvert^2}$	photonics

- A very general class of encodings can be nicely captured in Fourier space, but I won't discuss that now.

QUANTUM REPRESENTERS

- Now we can apply the Representer Theorem to learn what optimal quantum models look like:

$$f = \sum_m \alpha^m \lambda_{x^m} = \sum_m \alpha^m \langle \cdot \rangle_m$$

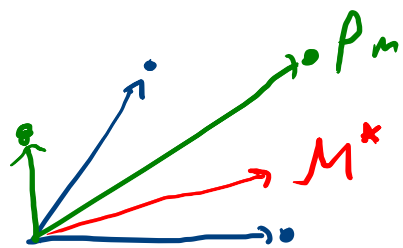
for $x^m \in \mathcal{X}_T$. This is a little abstract.

- Really, this just means optimal measurements are built from density matrices ρ_m of training data:

$$\begin{aligned} f(x) &= \sum_m \alpha^m \langle \rho_x \rangle_m \\ &= \sum_m \alpha^m \operatorname{tr}[\rho_m \rho_x] \\ &= \operatorname{tr}\Big[\sum_m \alpha^m \rho_m \, \rho_x\Big] \\ &= \operatorname{tr}\big[\mathcal{M}^* \rho_x\big]. \end{aligned}$$

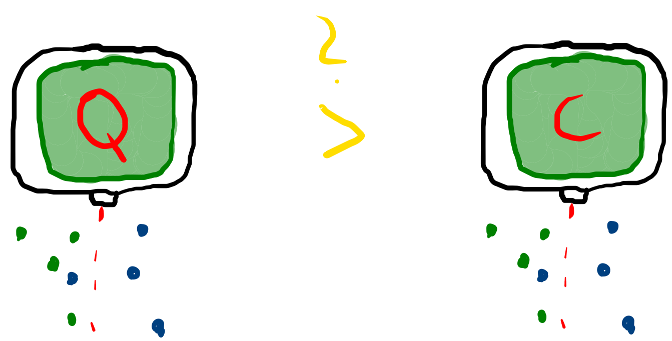
SO WHAT?

- The take-home message is that classical kernel theory gives us optimal quantum models. This is cool!



$$M^* = \sum_m \alpha^m \rho_m$$

- This is interesting from a theory perspective, but also practical:
  - it tells us what to optimize, i.e. the α^m, and
  - what controls the scaling, i.e. M, the size of $\mathcal{X}_T$.
- Question: does this translate into a quantum speedup?



LOSS AND OPTIMIZATION

- The regularized empirical risk $\hat{R}$ takes the form

$$\hat{R}(f) = \frac{1}{M} \sum_m L[x^m, y^m, f(x^m)] + g(\|f\|).$$

loss function   regularizer

- For $g = \lambda \|f\|^2$ and a candidate optimal quantum model f^*,

$$\begin{aligned}\|f^*\|^2 &= \sum_{m m'} \alpha_m \alpha_{m'} \operatorname{tr}[\rho_m \rho_{m'}] \\ &= \sum_{m m'} \alpha_m \kappa(x^m, x^{m'}) \alpha_{m'} \\ &= \vec{\alpha}^T K \vec{\alpha}.\end{aligned}$$

Gram matrix

- The optimal measurement is then

$$f_{opt} = \inf_{f^*} \left[\frac{1}{M} \sum_m L + \vec{\alpha}^T K \vec{\alpha} \right]$$

which is a convex optimization problem for α^m if L is convex, e.g. hinge loss $L = \max(0, 1 - f(x) y)$ for "quantum" SVM.

SPEEDUPS WITH QRAM

- There is a speedup for training qSVMs provided we can efficiently implement quantum Random Access Memory (qRAM):

$$U_{LOAD}: \; |j\rangle|0\rangle \mapsto |j\rangle|\mathrm{mem}(j)\rangle .$$

address register — $|j\rangle$; memory register — $|0\rangle$

- In particular, if the address register refers to training data, and loading takes $\mathcal{O}(\underbrace{n}_{\log M})$ steps, building the Gram matrix

$$K_{m,m'} = k(x^m, x^{m'})$$

takes $\mathcal{O}(M)$ steps. [For $x \in \mathbb{R}^N$, it's $\mathcal{O}(MN)$. This is the bottleneck.]

- In contrast, classical SVMs take $\mathcal{O}(M^2)$ steps to train. [A little more carefully, it is $\mathcal{O}(M^2N + M^3)$.]
- Thus, efficient qRAM, e.g. "bucket brigade" model, gives a quadratic speedup for training SVMs!

THANKS!

QUESTIONS?

1. "Supervised QML models are kernel methods" (2021), Schuld.
2. "QML in feature Hilbert spaces" (2018), Schuld & Killoran.
3. "Supervised learning with QC" (2018), Schuld & Petruccione.
4. "Quantum embeddings for ML" (2020), Lloyd et al.
5. "qSVMs for big data classification" (2014), Rebentrost, Mohseni, Lloyd.
6. "qRAM" (2007), Giovannetti, Lloyd & Maccone.