Kernel Methods: applications, promises and open issues

Jean-Marc Mercier

MPG-Partners

25 06 2024

Introduction - Plan presentation

Kernel Methods: applications, promises and open issues...

- Predictive machine with uncertainty quantification.
- Partial differential equations (PDE) - mathematical physics.
- Statistic algorithms, Generative methods, Reinforcement Learning.
- Feed-back on kernel methods.

Some related keywords

- **Supervised learning**, **Clustering** methods. **Large dataset**.
- **Mesh-free methods, Particle methods**.
- **Encoders / Decoders**, **Actor Critic**, **Q-Learning**.

RKHS stands for Reproducing Kernel Hilbert Space.

Predictive machine and uncertainty quantification

Definition : a predictive (or learning) machine is an algorithm (m = neural nets, trees, ..., kernels) defining a function of $z \in \mathbb{R}^D$

$$z \mapsto f_m(z, \theta) \in \mathbb{R}^{D_f} \text{ is called the prediction}$$

Fitting a predictive machine

to a training set $(X, f(X)) \in \mathbb{R}^{N_X, D+D_f}$ corresponds to solving for some norm $\|\cdot\|$

$$\inf_{\theta} \|f_m(\cdot, \theta) - f(\cdot)\|$$

error quantification corresponds to find an error bound of kind $\|f_m(\cdot, \theta) - f(\cdot)\|$. A classical example: $\|f_m(Z, \theta) - f(Z)\|_{\ell^2}$, $Z \in \mathbb{R}^{N_Z, D}$, $f(Z) \in \mathbb{R}^{N_Z, D_f}$ are reference (ground truth) values ($\mapsto$ cross validation).

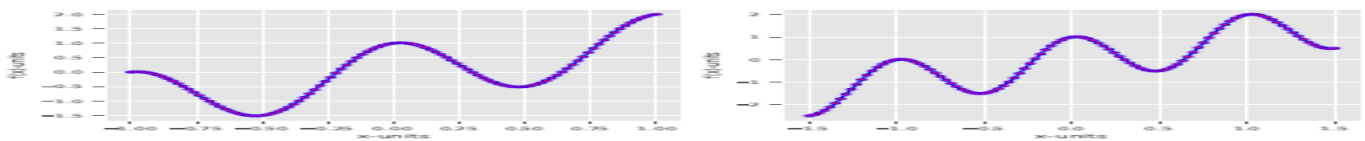


Figure 1: left training set (X,f(X)), right test set (Z,f(Z)).

Reminder on kernels

A kernel is a symmetrical function $k(x, y)$

- **Translation invariant** kernels $k(x, y) = f(x - y)$: eg Gaussian $k(x, y) = \exp(-|x - y|^2)$.
- **Polynomial features map** kernels $k(x, y) =< f(x), f(y) >^{\alpha}$.
- **Kernel engineering** $k(x, y) = (k_1 + k_2)(x, y)$, $k(x, y) = (k_1 \times k_2)(x, y)$, $k(x, y) = (k_1 * k_2)(x, y)$.
- **Vector or functional kernels** $|(k_1, k_2, \cdots)(x, y)|^2$ (vector), $k^*(x, y) := |x * y|^{\alpha}$ (convolutional), $\hat{k}(x, y) = k(\hat{x}, \hat{y})$ (Fourier type).

Gram matrix $k(X, Y) := \left(k(x^i, y^j)\right)_{i,j} \in \mathbb{R}^{N_X, N_Y}$.

Definite positive kernels: $k(X, X)$ is s.d.p if $x^i \neq x^j$, $i \neq j$.

A kernel always needs a normalization map $k(x, y) \equiv k(S(x), S(y))$

Kernel distances : $d(x, y) = k(x, x) + k(y, y) - 2k(x, y)$, $d(X, Y) = \ldots$

Predictive kernel machine

A kernel predictive machine is defined as $f_k(z, \theta) := k(z, Y)\theta, \quad \theta \in \mathbb{R}^{N_Y, D_f}$

Fitting a kernel predictive machine to a training set $X, F(X)$:

$$f_k(z, \theta) := k(z, Y)\theta, \quad \theta := \Big(K(X, Y) + \epsilon R(X, Y)\Big)^{-1} f(X)$$

Kernel quantification error

$$\|f_k(Z, \theta) - f(Z)\|_{\ell^2} \leq \Big(d(Z, Y) + d(X, Y)\Big)\|f\|_{\mathcal{H}_k}$$

Reproducibility properties

$$\|f_k(X, \theta) - f(X)\|_{\ell^2} \leq d(X, X)\|f\|_{\mathcal{H}_k} = 0, \quad (Y = X, \epsilon = 0)$$

Kernel differential operators

$$\nabla f_k(z, \theta) = \nabla_z k(z, Y)\theta$$

Issues with predictive kernel machines

For extrapolation, the parameters set θ is fit according to $\theta := K(X,X)^{-1}f(X)$.

algorithmic complexity and memory storage : N_X^2 (Gram matrix) plus N_X^3 (Invertion) operations: can't handle **large dataset**.

Clustering methods

$$\|f_k(X,\theta) - f(X)\|_{\ell^2} \leq d(X,Y)\|f\|_{\mathcal{H}_k}, \quad Y^* = \arg\inf_Y d(X,Y)$$

However the problem $\inf_Y d(X,Y)$ is still computationally intensive. Heuristic : k-means type algorithm with kernel distances can be quite efficients.

Low-rank approximations - Nystrom methods

$$Y^* = \arg\inf_{Y\subset X} \|I_d - k(X,Y)k(Y,Y)^{-1}k(Y,X)\|$$

Still require $N_X^2 N_Y + N_Y^3$ operations, but the following ersatz is very efficient

$$Y^* = \arg\inf_{Y\subset X} \|f(X) - K(X,Y)K(Y,Y)^{-1}f(Y)\|$$

Kernel methods for Partial Differential Equations

Kernel methods are very handy for PDE:

- **Kernel operators.** Kernels allows to define easily complex differential operators and to manipulate them.
- **Meshless methods.** No need to handle a mesh, making the code easier.
- **Particle methods.** The mesh can move, redefining its differential operators. Very efficient for **Lagrangian** methods.
- **Boundary conditions.** Kernel interpolation operators allows to take into account easily complex boundary conditions.
- **Convergence analysis.** Provide tools to analyze the convergence and complexity.

However classical schemes (finite differences, finite elements) are more performing for low dimensions ($D = 1, 2$). Usage of kernel methods

- **PDE in high dimensions:** anytime.
- **PDE in low dimensions:** for fast prototyping, or parametric PDEs (solutions depending on extra parameters), or unstructured meshes.
- **To learn PDEs** : learn the solution of a PDE ($u^{n+1} = f_k(\theta^n, u^n)$) to fasten computations of classical schemes.

PDE examples

The SABR process solved in Lagrangian coordinates

$$dB_t = \sigma_t \left(B_t\right)^{\beta} \, dW_t, \quad d\sigma_t = \alpha \sigma_t \, dZ_t$$

Video: At each time, the scheme compute the best **Monte-Carlo** samples $X^*(t) := \inf_{X \in \mathbb{R}^{N,D}} |\mathbb{E}^{B_t}[f] - \frac{1}{N} \sum_{n=1}^{N} f(x^n)|$. Alternative to **low-discrepancy** sequences.

Navier-Stokes equations in Lagrangian coordinates

Video: A viscous fluid running into a pipe (null normal component velocity boundary conditions).

Kernels encoders and decoders

Consider X, Y two set of points and consider the predictive machine

$$z \mapsto k(z, X)\theta, \quad \theta := k(X, X)^{-1} Y$$

defining a mapping from $\mathbb{R}^{D_X}$ to $\mathbb{R}^{D_Y}$. Can we invert it ?

Homogenous case ($D_X = D_Y = D$): find a permutation σ

$$\sigma^* := \arg\inf_{\sigma} \sum_n d(x^n, Y[\sigma^n]), \quad \theta^* := k(X, X)^{-1} Y[\sigma^*]$$

Then $z \mapsto k(z, X)\theta^*$ is invertible (**optimal transport**)

Inhomogenous case ($D_X \neq D_Y$) : find a permutation σ such that

$$\sigma^* := \arg\inf_{\sigma} |\sum_n \nabla_k Y[\sigma^n])|^2, \quad \theta^* := k(X, X)^{-1} Y[\sigma^*]$$

Generalized travelling salesman problem. Drawback : computationally intensive.

Applications: generative methods

Let $\mathbb{X}, \mathbb{Y}$ be **any** two probability measures, taking values into $\mathcal{X} \subset \mathbb{R}^{D_X}$,$\mathcal{Y} \subset \mathbb{R}^{D_Y}$.

Figure 2: Generating high res faces using low res cats

- $\mathcal{L} : \mathcal{Y} \mapsto \mathcal{X}$ is the **encoder**. Example: faces res. $D_Y = 120000$.
- $\mathcal{X}$ is the **latent** space. example: cats res. $D_X = 100$.
- The inverse map $\mathcal{L}^{-1} : \mathcal{X} \mapsto \mathcal{Y}$ is the **decoder**.
- The projection operator $z \mapsto (\mathcal{L}^{-1} \circ \mathcal{L})(z)$ is called a **reconstruction**

Alternative approaches: GAN, WGAN, ...

Applications of kernel generative methods (KGM)

The sampler function : reproduce a random variable

Generation of a variate, that is **statistically** coherent with historical observations.

Figure 3: Generated (right) images of CelebA dataset, matched to the original CelebA dataset (left).

Other applications of KGM.

Conditioned random variables

Consider $\mathbb{X} \in \mathbb{R}^{N,D_X}, \mathbb{Y} \in \mathbb{R}^{N,D_Y}$ two variates of dependent random variables. Consider $\mathbb{Z} = \Big(\mathbb{X},\ \mathbb{Y}\Big) \in \mathbb{R}^{N,D_X+D_Y}$ the joint random variable. We can approximate the following conditioned distribution:

$$\mathbb{Y}_X := \mathbb{E}\Big(\mathbb{Y}|x\Big), \quad x \in \text{supp}(\mathbb{X})$$

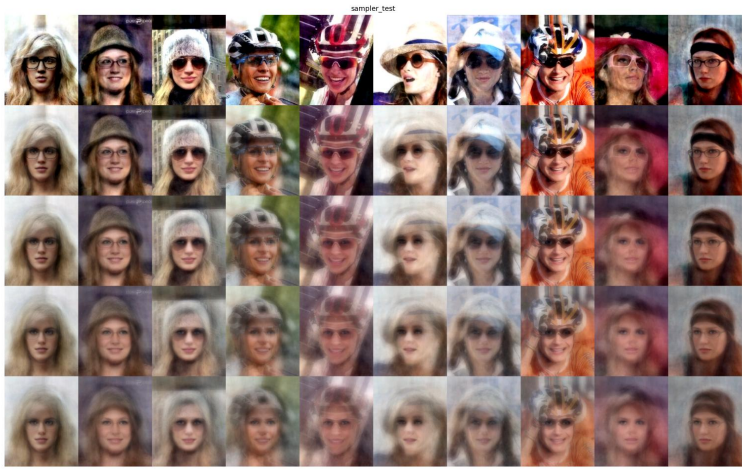


Figure 4: Removing hat and glasses of CelebA images.

Time series prediction with generative methods.

Real market data, retrieved from January 1, 2016 to December 31, 2021, for three assets: Google, Apple and Amazon.

Figure 5: charts for Apple Amazon Google

Format : N_X is the number of observed paths. Here $N_X = 1$. D_X is the number of underlyings. Here $D_X = 3$. T_X is the number of times buckets. Here $T_X = 506$ (2Y).

$$X = \left(x_d^{n,k}\right)_{d=1\ldots D}^{n=1\ldots N_X, k=1,\ldots,T_X} \in \mathbb{R}^{N_X, D_X, T_X}.$$

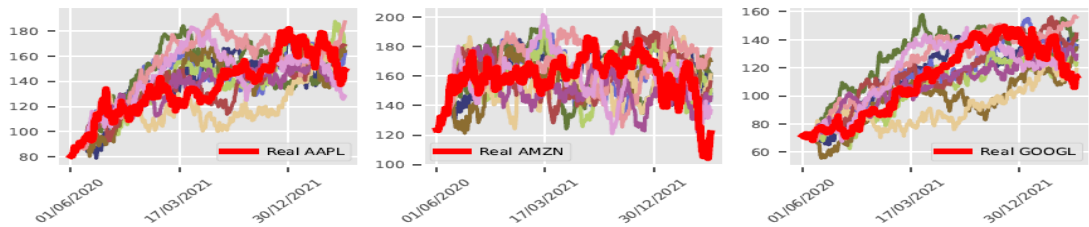
Agnostic models.

We call a **free model**, or *agnostic model* the following framework for time series

$$F\Big(X\Big) = \varepsilon$$

- $\varepsilon \in \mathbb{R}^{N_\epsilon, D_\epsilon, T_\epsilon}$ is a white, or conditioned noise.
- $F : \mathbb{R}^{N_X, D_X, T_X} \mapsto \mathbb{R}^{N_\epsilon, D_\epsilon, T_\epsilon}$ is a **invertible** continuous map.
- **Incentive:** $X = F^{-1}\Big(\varepsilon\Big)$ is one iid path of the model (reproducibility).
- **Path Generation:** easy Monte-Carlo sampling with generative methods ($Z = F^{-1}\Big(\eta\Big)$).

All stochastic processes can be re-interpreted as agnostic models. For instance, the multi-time steps model ARMA(p,q)



Extending models with conditioning $F(X) = \mathbb{E}(\epsilon|Y)$.

Promises and open issues.

- Versatile, universal approach to mathematical physic.
- Very performing also for statistic and machine learning, specially in the small to medium size dataset regime.
- Interesting for industrial purposes. Reproducibility and error bounds are important for audit purposes. Fast learning is paramount to reduce energy consumption.

Kernel methods should be mainframe either for industry and research. But:

- Still migrating from research (large dataset).
- **No standard library dedicated to kernel.** Kernel software is very fragmented, no consensual approach (scikit-learn, tensorflow, keops, pytorch, codpy, ...).
- Few available resources to develop kernel methods.

Some references

- CodPy: a Python library for numerics, machine learning, and statistics : https://arxiv.org/pdf/2402.07084.
- Some names to complete a book shelves with kernel methods : Thomas-Agnan, Berlinet, Smolla, Scholkopf, Greton, Fasshauer, ...