Feature Maps, Kernels and Regularization

Supervised Learning (COMP0078)

Carlo Ciliberto

AI Centre, Department of Computer Science
University College London

In This Class

1. Beyond Linear Models: Feature Maps
2. Computational Considerations
3. Dual Solution
4. Implicit Feature Maps
5. Kernels
6. Regularization

Part 1: Beyond Linear Models

Recap: Training by fitting some data

The first example of training algorithm:

- Given a dataset of samples, and $(x_i, y_i)_{i=1}^n$
- A loss $\ell(z, y)$ measuring the "error" when predicting z instead of y,

Goal. Find a function $f : \mathcal{X} \to \mathcal{Y}$ minimizing

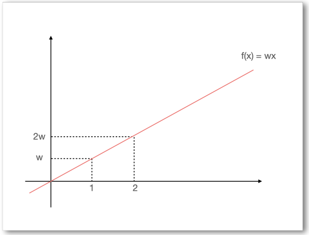
$$\frac{1}{n}\sum_{i=1}^{n} \ell(f(x), y)$$

Problem. we need[1] to choose a family of candidate models for f!

[1] cannot minimize over all f? **why?**

Recap: Linear Models

Question. What is the simplest form of function we can learn?



For $\mathcal{X} \subset \mathbb{R}^d$ a vector space, we can consider **linear models**

$$f : \mathcal{X} \to \mathbb{R} \qquad \text{such that} \qquad f(x) = w^\top x$$

for any $x \in \mathcal{X}$ **input vector** and $w \in \mathbb{R}^d$ the **parameter** vector of f.

Recap: Least Squares Minimization

We then considered $\ell(z, y) = (z - y)^2$ the squared loss and learned

$$\hat{w} = \underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \ \frac{1}{n} \sum_{i=1}^{n} (w^\top x_i - y_i)^2$$

by:

- Writing the objective in matrix notation $L(w) = \frac{1}{n} \|Xw - y\|^2$
- Setting the derivative $\nabla_w L(w) = 2\frac{1}{n} X^\top X w - X^\top y = 0$
- solving the corresponding linear system $X^\top X \hat{w} = X^\top y$ yielding

$$\hat{w} = (X^\top X)^{-1} X^\top y$$

What if Linear Models are not Enough?

This is all good and well if x and y are linearly related.

But what if they are **not**?

Polynomials. Let us start with the natural non-linear generalization.

Let $\mathcal{X} = \mathbb{R}$ we model $f : \mathcal{X} \to \mathbb{R}$ as a polynomial of degree $p \geq 0$

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_p x^p$$

where $a_0, \ldots, a_p \in \mathbb{R}$ are the coefficients.

Polynomial Least Squares

Then, learning a polynomial to fit a training dataset becomes

$$\min_{a_0,\dots,a_p} \frac{1}{n} \sum_{i=1}^{n} (a_0 + a_1 x_i + a_2 x_i^2 + \cdots + a_p x_i^p - y_i)^2$$

(!) But the terms x_i^q for $q = 0, \dots, p$ are “just” numbers!

The optimization problem is still **linear** with respect to the parameters a_q

The Polynomial “Feature map”

Define $\phi : \mathcal{X} \to \mathbb{R}^{p+1}$ such that $\phi(x) = (1, x, x^2, \dots, x^p)^\top$

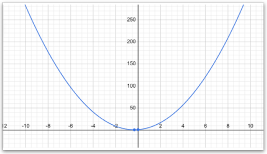
“Rename” the parameters $w = (w_1, \dots, w_{p+1}) = (a_0, \dots, a_p)$.

Then we can model the polynomial f as

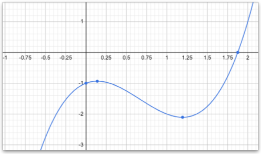
$$f(x) = w^\top \phi(x) \tag{1}$$

Namely a linear model, in the “embedded” space $\phi(\mathcal{X})$

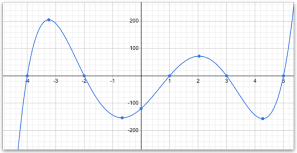
Examples



$p = 2$



$p = 3$



$p = 5$

Polynomial Least Squares (Continued)

Optimization Problem. The training objective becomes

$$\frac{1}{n}\sum_{i=1}^{n}(w^\top \phi(x_i) - y_i)^2 = \|\Phi w - y\|^2$$

Where $\Phi \in \mathbb{R}^{n\times p}$ is the input matrix, with i-row corresponding to the **embedded** point $\phi(x_i)$ (taking the place of $X \in \mathbb{R}^{n\times d}$.

Solution. Analogously to the linear case, the objective is **minimized** by

$$\hat{w} = (\Phi^\top \Phi)^{-1}\Phi^\top y$$

Same optimization, but we have replaced x with $\phi(x)$!

Basis Functions / Feature Maps

This construction can be generalized:

- First, to polynomials of degree p over vectors $x \in \mathbb{R}^d$

$$\phi(x) = (x_1^{j_1} x_2^{j_2} \cdots x_d^{j_d})_{j_1, \ldots, j_d \in J_p}$$

  with indices spanning $J_p = \{(j_1, \ldots, j_d) \ s.t. \ \sum_{i=1}^d j_i = p\}$

- Then, to any $\phi : \mathcal{X} \to \mathbb{R}^D$ for some "latent" dimension D.

Code-wise

One appealing advantage of this perspective is **modularity**:

- The logic to **learn** the solution $\hat{w}$

```
import numpy as np

X,y = load_dataset()    # load pairs of input and outputs
Phi = extract_features(X)

hat_w = np.linalg.solve(Phi.T @ Phi, Phi.T @ y)   # "learn"
```

- is separated from that of embedding the input $x \mapsto \phi(x)$

```
# Don't do anything
def extract_linear_features(X):
    return X

# Assume scalar inputs
def extract_quadratic_features(X):
    return np.concatenate((np.ones_like(X),X, np.power(X,2)),
                                                        axis=1)
```

Part 2: Computations

The Cost of Minimizing the Squared Loss

We have a method for solving the squared loss.

Question. How efficient is it?

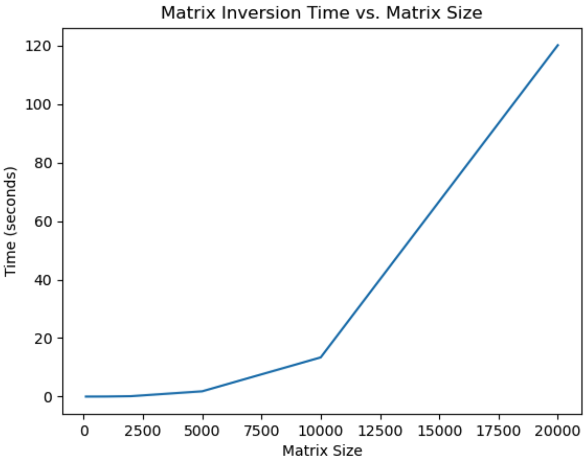
In other words...

how much time (and memory) would it take to obtain $\hat{w}$?

Answer. It depends:

- On the machine
- On the training size n
- On the number D of input dimensions.

For example, on my machine...



Costs in Big O notation

Let's try to be more rigorous in asking the question...

Question. What is the cost in **Big O** notation?

Answer:

- $O(D^3 + nD^2)$ in time[2],
- $O(D^2 + Dn)$ in memory

[2]We are referring to standard methods implemented in BLAS/LAPACK

Aside: Big O notation

Informally, $f(x) = O(g(x))$ if it behaves "essentially" like $g(x)$.

Formally[3]**,** there exists $C > 0$ and x_0 such that for any $x > x_0$

$$|f(x)| < Cg(x)$$

Note. two functions with "same" big O behavior might differ a lot by:

- The "activating" x_0 (one might behave like g(x) a lot sooner),
- The constant C (one might have a much larger constant).

Hence two methods with same big O, might still be quite different in practice in terms of time and memory.

[3]Big O notation will be discussed in detail in the TA sessions.
I recommend attending if you are not familiar with it!

When is D too large?

Let assume is fixed, say $n = 1000$. How large can we take D?

Ideally, as large as possible[4]!

In practice, not too much: cost grows cubically in D.

Examples. Polynomials $f(x)$ of degree p over d variables ($\mathcal{X} = \mathbb{R}^d$):

- $d = 1, \quad p = 3 \Longrightarrow D = 4$
- $d = 3, \quad p = 5 \Longrightarrow D = 56$
- $d = 5, \quad p = 10 \Longrightarrow D = 3003$
- $d = 10, p = 15 \Longrightarrow D = 184756$

[4]Well... not really because of overfitting (we will get back to this point), but we would not like computations to be our bottleneck!

A Closer Look

Where is the cost $O(D^3 + nD^2)$ coming from?

$$\hat{w} = (\underbrace{\Phi^T\Phi}_{\substack{\text{Multiplication}\\ (D\times n)\cdot(n\times D)\to O(nD^2)}})^{\overset{\substack{\text{Inversion}\\ D\times D\to O(D^3)}}{\boxed{-1}}}\Phi^\top y$$

Is there any way we could estimate $\hat{w}$ faster?

Part 3: Dual Solution

Alternative Formulation for Squared Loss Minimization

We will manipulate the solution

$$\hat{w} = (\Phi^\top \Phi)^{-1} \Phi^\top y$$

to derive:

- an **equivalent** characterization
- that can be **faster** in some regimes.

Our Strategy

There are many ways to approach this problem.

We will use a strategy that makes significant use of **linear algebra**.

This will familiarize us with tools that we will use also in the future.

Singular Value Decomposition

We will start by considering the **Singular Value Decomposition (SVD)**

$$\Phi = U\Sigma V^\top$$

Where:

- $r = \min(n, D)$
- $U \in \mathbb{R}^{n\times r}$ and $V \in \mathbb{R}^{D\times r}$ have **orthonormal** columns

$$U^\top U = I \qquad \text{and} \qquad V^\top V = I$$

- $\Sigma \in \mathbb{R}^{r\times r}$ is a **diagonal** matrix with only non-negative entries.
- The diagonal vector $\sigma = \operatorname{diag}(\Sigma) \in \mathbb{R}^r$ is the **spectrum** of Σ
- The number of non-zero entries in σ is the rank of Φ.

The SVD and other useful prerequisites will be discussed also in the TA sessions. Attendance is recommended.

SVD and Squared Loss Minimizer

(!) For Simplicity. Assume Φ "full rank": $\mathrm{rank}(\Phi) = r$.

Let us plug in the SVD of $\Phi = U\Sigma V^\top$ into $\hat{w}$

$$\hat{w} = (\Phi^\top\Phi)^{-1}\Phi^\top y = (V\Sigma U^\top U\Sigma V^\top)^{-1}V\Sigma U^\top y$$

- Since $U^\top U = I$,we have $V\Sigma U^\top U\Sigma V^\top = V\Sigma^2 V^\top$
- Since $V^\top V = I$, we have $(V\Sigma^2V^\top)(V\Sigma^{-2}V^\top) = I$. Hence

$$(V\Sigma^2V^\top)^{-1} = V\Sigma^{-2}V^\top$$

- Combining the two

$$\hat{w} = (V\Sigma^{-2}V^\top)V\Sigma U^\top y = V\Sigma^{-1}U^\top y$$

SVD and Squared Loss Minimizer (Continued)

We can now "go back" by making the pairs $U^\top U$ and $V^\top V$ **"appear"**

$$\begin{aligned}
\hat{w} &= V\Sigma^{-1}U^\top y \\
&= V\Sigma\Sigma^{-2}U^\top y \\
&= V\Sigma U^\top U\Sigma^{-2}U^\top y \\
&= \underbrace{(V\Sigma U^\top)}_{\Phi^\top}(U\Sigma^{-2}U^\top)y \\
&= \Phi^\top(\underbrace{U\Sigma^2U^\top}_{\Phi\Phi^\top})^{-1}y \\
&= \Phi^\top(\Phi\Phi^\top)^{-1}y
\end{aligned}$$

SVD and Squared Loss Minimizer (Continued)

We can now "go back" by making the pairs $U^\top U$ and $V^\top V$ **"appear"**

$$\begin{aligned}
\hat{w} &= V\Sigma^{-1}U^\top y \\
&= V\Sigma\Sigma^{-2}U^\top y \\
&= V\Sigma U^\top U\Sigma^{-2}U^\top y \\
&= \underbrace{(V\Sigma U^\top)}_{\Phi^\top}(U\Sigma^{-2}U^\top)y \\
&= \Phi^\top(\underbrace{U\Sigma^2U^\top}_{\Phi\Phi^\top})^{-1}y \\
&= \Phi^\top(\Phi\Phi^\top)^{-1}y
\end{aligned}$$

(!) Note in both this and the previous slide we have been a bit sloppy about matrix inversion. We will get back to this soon, to make our derivation more rigorous.

Dual vs "Primal" formulation

After a bit of work we found out that $\hat{w}$ can be written as

$$\hat{w} = \Phi(\Phi^{-1}\Phi)^{-1}y$$

...and... so what?

Dual vs "Primal" formulation

After a bit of work we found out that $\hat{w}$ can be written as

$$\hat{w} = \Phi(\Phi^{-1}\Phi)^{-1}y$$

...and... so what?

At a superficial look, this formula is not different from the **"primal"**

$$\hat{w} = (\Phi^\top\Phi)^{-1}\Phi^\top y$$

We just **"swapped"** $\Phi^\top$ and $(\Phi^\top\Phi)^{-1}$...

...or did we?

The Benefits of the Dual Formulation

$$(\Phi^\top\Phi)^{-1}\Phi^\top y \qquad \text{Vs} \qquad \Phi(\Phi^{-1}\Phi)^{-1}y$$

The matrices inverted have different dimensions!

- $\Phi^\top\Phi$ is $D \times D$
- $\Phi\Phi^\top$ is $n \times n$.

Therefore the computational costs for the two formulations are:

- **Primal:** $O(D^3 + nD^2)$
- **Dual:** $O(n^3 + Dn^2)$

Therefore in the Dual formulation the dominating factor is n!
(wrt which cost grows cubically)

The Benefits of the Dual Formulation (Continued)

Of course, this does not mean that the dual formulation is “better”

It simply shows that **when** $N < D$ the dual formulation is faster.

This is appealing, however, since we have seen (e.g. the polynomials) that the feature space can **grow** pretty quickly wrt the problem parameters!

In particular, it will be critical when we will consider feature space that are infinite dimensional!

Dotting our "i"s and crossing our "t"s: (Pseudo)inversion

We have been sloppy in our use of the notation $(\Phi^\top\Phi)^{-1}$ and $(\Phi\Phi^\top)^{-1}$.

If $n \neq D$ however, at least one of the two matrices will not be invertible.

More more generally, if $\operatorname{rank}(\Phi) < \min(n, D)$, neither will.

The correct quantity that we should have used is the **Pseudoinverse**.

Def. The pseudoinverse[5] of $\Phi \in \mathbb{R}^{n\times D}$ is a matrix $\Phi^\dagger \in \mathbb{R}^{D\times n}$ such that

- Behaves like a "weak inverse":

$$\Phi\Phi^\dagger\Phi = \Phi \qquad \text{and} \qquad \Phi^\dagger\Phi\Phi^\dagger = \Phi^\dagger,$$

- Both $\Phi\Phi^\dagger$ and $\Phi^\dagger\Phi$ are Symmetric.

[5]can be generalized to Complex matrices.

Pseudoinverse and SVD

The SVD of the pseudoinverse of $\Phi = U \Sigma V^\top$ is

$$\Phi^\dagger = V \Sigma^\dagger U^\top$$

Where $\Sigma^\dagger$ is the diagonal matrix with entries

$$(\Sigma^\dagger)_{ij} = \begin{cases} \frac{1}{\Sigma_{ij}} & \text{if } \Sigma_{ij} \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

Dotting our "i"s and crossing our "t"s: (Pseudo)inversion

We can therefore retrace our derivation of the dual formulation[6] to obtain

$$\hat{w} = \underbrace{(\Phi^\top \Phi)^\dagger \Phi^\top y}_{Primal} = \underbrace{\Phi^\top (\Phi \Phi^\top)^\dagger y}_{Dual}$$

which is now well-defined.

[6] Reccomendation: do that as an exercise!

Code-wise

In practice... Pseudoinverse calculation is often numerically unstable[7] (we will get back to this later).

It is recommended to use other methods. For example **linear solvers** [8]

Primal

```
hat_w = np.linalg.solve(Phi.T @ Phi, Phi.T @ y)
```

Dual

```
alpha = np.linalg.solve(Phi @ Phi.T, y)    # dual coefficients
w_hat = Phi.T @ alpha
```

[7]meaning you might get the wrong $\hat{w}$ out!

[8]Namely routine for solving problem of the form $Ax = b$, which is indeed $x = A^{\dagger}b$

Part 4: Implicit Feature Maps

A Deeper Look into the Dual Formulation

The code snippet for the **dual** formulation

```
alpha = np.linalg.solve(Phi @ Phi.T, y)    # dual coefficients
w_hat = Phi.T @ alpha
```

highlights an interesting structure.

Denote $K = \Phi\Phi^\top$, then the solution $\hat{w}$ is

$$\hat{w} = \Phi^\top \alpha \qquad \text{where} \qquad \alpha = K^{-1} y$$

In some sense, we have **decoupled**:

- The "training" (the computation intensive inversion of α).
- The evaluation $\hat{f}(x) = \phi(x)^\top \hat{w} = \phi(x)\Phi^\top \alpha$.

(?) Why is this interesting?

Inner Products

We have two interesting observations:

1. $K_{ij} = (\Phi\Phi^\top)_{ij} = \phi(x_i)^\top \phi(x_j)$,
2. Evaluating the learned function $\hat{f}$ in a new point x yields

$$\hat{f}(x) = \phi(x)^\top \hat{w} = \phi(x)\Phi^\top \alpha = v(x)^\top \alpha$$

where we have defined the **evaluation vector** $v(x) \in \mathbb{R}^n$

$$v(x) = (\phi(x)^\top \phi(x_1), \ldots, \phi(x)^\top \phi(x_n))^\top$$

Inner Products

We have two interesting observations:

1. $K_{ij} = (\Phi\Phi^\top)_{ij} = \phi(x_i)^\top \phi(x_j)$,

2. Evaluating the learned function $\hat{f}$ in a new point x yields

$$\hat{f}(x) = \phi(x)^\top \hat{w} = \phi(x)\Phi^\top \alpha = v(x)^\top \alpha$$

where we have defined the **evaluation vector** $v(x) \in \mathbb{R}^n$

$$v(x) = (\phi(x)^\top \phi(x_1), \ldots, \phi(x)^\top \phi(x_n))^\top$$

All the operations required to train and evaluate $\hat{w}$ depend only on the inner products $\phi(x)^\top \phi(x')$!

Are Inner Products Better

Question. Are there feature maps ϕ for which we can compute $\phi(x)^\top \phi(x')$ "directly" from x and x'...

...without explicitly needing the embedded vectors?

Are Inner Products Better

Question. Are there feature maps ϕ for which we can compute $\phi(x)^\top\phi(x')$ "directly" from x and x'...

...without explicitly needing the embedded vectors?

Yes!

Example. Consider $x \in \mathcal{X} \subset \mathbb{R}^2$ and $\phi : \mathbb{R}^2 \to \mathbb{R}^3$, hence $D = 3$

$$\phi(x) = (\ x_1^2\ ,\ \sqrt{2}x_1x_2\ ,\ x_2^2\ )^\top$$

In principle $\phi(x)^\top\phi(x')$ requires the sum of $D = 3$ products. However...

$$\begin{aligned}\phi(x)^\top\phi(x') &= \underbrace{x_1^2x_1'^2}_{a^2} + 2\underbrace{x_1x_1'}_{a}\underbrace{x_2x_2'}_{b} + \underbrace{x_2^2x_2'^2}_{b^2} \\ &= (a+b)^2\end{aligned}$$

...one sum and one product are **sufficient!**

Spaces of Polynomials and Inner Product

The example generalizes to polynomials of degree p in dimension d:

Proposition. For any $p, d \in \mathbb{N}$ there exist coefficients $C_{j_1,\dots,j_d} \in \mathbb{R}$ and a feature map $\phi : \mathbb{R}^d \to \mathbb{R}^D$

$$\phi(x) = \left(C_{j_1,\dots,j_d}\ x_1^{j_1} \cdots x_d^{j_d}\right)_{\sum_i j_i \le p}$$

such that

$$\phi(x)^\top \phi(x') = (x^\top x' + 1)^p$$

Space of Polynomials and Inner Product

A few observations:

- The $C_{j_1,\ldots,j_d}$ can be obtained by expanding the product $(x^\top x' + 1)^p$
- $D = O(p^d)$ ($D =$ the number of addends in the expansion above).
- Evaluating $(x^\top x' + 1)^p$ requires "only" $O(d)$ operations.

[9]Another, non-trivial advantage: we **don't need** to calculate the $C_{j_1,\ldots,j_d}$ explicitly!

Space of Polynomials and Inner Product

A few observations:

- The $C_{j_1,\dots,j_d}$ can be obtained by expanding the product $(x^\top x' + 1)^p$
- $D = O(p^d)$ ($D =$ the number of addends in the expansion above).
- Evaluating $(x^\top x' + 1)^p$ requires "only" $O(d)$ operations.

Therefore, the computational costs are:

- **Primal.** $O(p^{3d} + np^{2d})$
- **Dual.** $O(n^3 + p^d n^2)$
- **Inner Product + Dual.** $O(n^3 + dn^2)$

Therefore Dual formulation + Using only inner product provides a huge computational[9] advantage!

[9] Another, non-trivial advantage: we **don't need** to calculate the $C_{j_1,\dots,j_d}$ explicitly!

Kernels

Wishlist. More generally, we would like to find feature maps ϕ for which there exists a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that

$$k(x, x') = \phi(x)^\top \phi(x')$$

and the calculation **does not** require explicit knowledge of $\phi(x)$

Then. If evaluating $k(x, x')$ can be done in $O(\kappa)$ operations...

...it follows that learning $\hat{w}$ requires

$$O(n^3 + \kappa n^2)$$

with the **Inner Product + Dual** formulations[10]

[10]Recall that you will still need $O(n)$ operations to evaluate $f(x)$!

Inverting the Question

Now, consider our current question. . .

Can we find a feature map ϕ whose associated inner product k that can be evaluated without knowledge of $\phi(x)$?

. . . and let us turn it on its head by asking ourselves:

When does a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ correspond to the inner product of some feature map ϕ?

Why might this be interesting?

Searching for an Inner Product

Intuition. Inner products are a measure of similarity (actually correlation) between two objects. If $\phi(x)^\top \phi(x')$ is:

- **Positive and large.** Then x and x' are "similar" according to ϕ.
- **Close to zero.** Then x and x' are uncorrelated (not very similar)
- **Negative and large.** Then x and x' are oppositely correlated.

Sometimes it might be easier[11] to describe a desirable notion of similarity/correlation as a function $k(x, x')$ explicitly, rather than implicitly by designing a feature map ϕ.

BUT then, the question is whether such function is an inner product!

[11]another advantage: by designing k, we could choose one that is fast to compute!

Building Inner Product Maps

So our next question will be

Can we find a criterion to determine when a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ corresponds to the inner product of some feature map ϕ?

Better yet, we look for some constructive tools

Can we find some rules that allow us to build functions $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ that guarantee they correspond to the inner product of some feature map ϕ?

Part 5: Kernels

(Implicitly) Recovering Inner Products

Goal. Find an **alternative** characterization for an inner product function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ that **does not require** explicit knowledge of ϕ.

How? Let's look at some properties of inner product functions.

Observation. Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. For any set of points $(x_i)_{i=1}^n$, let $K \in \mathbb{R}^{n \times n}$ be the corresponding "Gram" matrix with entries

$$K_{ij} = k(x_i, x_j).$$

Then.[12] k is an inner product $\Longrightarrow$ K is Positive Semidefinite[13] $(K \succeq 0)$.

[12]Exercise! (Actually... we already proved some slides ago)
[13]This holds for any Gram matrix K (namely any $n \in \mathbb{N}$ and any set $(x_i)_{i=1}^n$)

Positive Semidefinite Gram Matrices

Recall. A matrix $K \in \mathbb{R}^{n\times n}$ is **Positive Semidefinite (PSD)** if it is symmetric and $v^\top K v \geq 0$ for any $v \in \mathbb{R}^n$

Equivalently K is PSD $\iff$ it's SVD is of the form[14] $K = U\Sigma U^\top$.

Is it important for K to be PSD?

[14]Note: here $U = V$ and recall that Σ has only non-negative entries

(Implicitly) Recovering Inner Products

Question. Is the converse "k is an inner product $\Longleftarrow K \succeq 0$" true?

Maybe. But is it important?

Strictly speaking, **it is not:** we can still define the estimator

$$\hat{f}(x) = v(x)^\top \alpha \qquad \text{with} \qquad \alpha = K^\dagger y$$

and $v(x) = (k(x, x_1), \ldots, k(x, x_n))^\top$ the "evaluation" vector.

[15] But if all this has not scared you, have a look at *reproducing kernel* **Krein** *spaces*

(Implicitly) Recovering Inner Products

Question. Is the converse "k is an inner product $\Longleftarrow K \succeq 0$" true?

Maybe. But is it important?

Strictly speaking, **it is not:** we can still define the estimator

$$\hat{f}(x) = v(x)^\top \alpha \qquad \text{with} \qquad \alpha = K^\dagger y$$

and $v(x) = (k(x, x_1), \ldots, k(x, x_n))^\top$ the "evaluation" vector.

However, this approach introduces a series of:

- **Numerical** issues in practice,
- **Methodological** issues algorithmically,
- **Statistical** issue theoretically.

In other words, it's a mess[15]!

[15]But if all this has not scared you, have a look at *reproducing kernel* **Krein** *spaces*

Positive Definite Kernels

So, let's assume that we **want** k to be PSD.

Def. *We say that* $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ *is a positive definite kernel if for any dataset* $(x_i)_{i=1}^n$ *the corresponding Gram matrix is PSD.*

So... is this enough for k to be an inner product?

If it Quacks Like a Duck...

Theorem. k is an inner product $\iff$ k is a positive definite kernel.

Yay!

In Depth. What can be shown is that there exist

- A **Hilbert** space $\mathcal{H}$ with **inner product** $\langle \cdot, \cdot \rangle_{\mathcal{H}}$
- A **feature map** $\phi : \mathcal{X} \to \mathcal{H}$

Such that

$$k(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{H}} \qquad \forall x, x' \in \mathcal{X}$$

Which is precisely what we wanted!

Aside: Hilbert Spaces

Recall. A **Hilbert** space is:

- A vector space (in our case we assume it to be real)
- Equipped with an **inner product** $\langle \cdot, \cdot \rangle_{\mathcal{H}}$
- ...and the corresponding **norm** $\| \cdot \|_{\mathcal{H}}$ such that $\|x\|_{\mathcal{H}}^2 = \langle x, x \rangle_{\mathcal{H}}$
- That is **complete**[16] with respect to $\| \cdot \|_{\mathcal{H}}$.

Recall. An **inner product** $\langle \cdot, \cdot \rangle$ over a (real) vector space $\mathcal{V}$ is a function $\langle \cdot, \cdot \rangle : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$ such that $\forall v, v', v'' \in \mathcal{V}$ and $\alpha, \beta \in \mathbb{R}$:

- **Symmetric** $\langle v, v' \rangle = \langle v', v \rangle$
- **(Bi)linear** $\langle \alpha v + \beta v', v'' \rangle = \alpha \langle v, v'' \rangle + \beta \langle v', v'' \rangle$
- **Positive definite** $\langle v, v \rangle > 0$

[16] that is, every Cauchy sequence admits a limit in $\mathcal{H}$.

Infinite Dimensional Hilbert Spaces

Note. when $\mathcal{H}$ is finite of dimension D, then $\mathcal{H} \cong \mathbb{R}^D$.

So far, we have only considered spaces of this form, but nothing prevents us to consider **infinite dimensional spaces**.

Example (The Gaussian Kernel) Let $\sigma > 0$. Then the function

$$k(x, z) = e^{-\|x-z\|^2/2\sigma}$$

is a kernel associated to an infinite Hilbert space (we will show this).

- Using **primal** or (naive) **dual** would be impossible ($D = +\infty$)!
- With **Kernel + Dual** it takes $O(n^3 + dn^2)$ ($k(x, z)$ requires $O(d)$).

The Gaussian Kernel Feature Map

Let's find a feature map for the Gaussian kernel.

For simplicity let $\sigma = 1$. Also, we will show[17] this for $\mathcal{X} = \mathbb{R}$.

- First, observe that by expanding the square

$$k(x, z) = e^{-(x-z)^2/2} = e^{-(x^2/2 - xz + z^2/2)} = e^{-x^2/2} e^{xz} e^{-z^2/2}$$

- Then, consider the Taylor series of the exponential for the mixed term

$$e^{xz} = \sum_{m=1}^{+\infty} \frac{(xz)^m}{m!} = \sum_{m=1}^{+\infty} \frac{x^m}{\sqrt{m!}} \frac{z^m}{\sqrt{m!}}$$

[17] The derivation becomes more involved for $\mathbb{R}^d$ with $d > 1$

The Gaussian Kernel Feature Map (Continued)

Let ℓ^2 denote the space of all (real) square-summable sequences[18]

Recall. ℓ^2 is a Hilbert space with $\left\langle (a_m)_{m=1}^{+\infty}, (b_m)_{m=1}^{+\infty} \right\rangle = \sum_{m=1}^{+\infty} a_m b_m$.

Define[19] $\phi : \mathbb{R} \to \ell^2$ the map such that $\forall x \in \mathbb{R}$

$$\phi(x) = \left(e^{-x^2/2} \frac{x^m}{\sqrt{m}} \right)_{m=1}^{+\infty}$$

Then,

$$\langle \phi(x), \phi(z) \rangle_{\ell^2} = \sum_{m=1}^{+\infty} e^{-x^2/2} \frac{x^m}{\sqrt{m}} \frac{z^m}{\sqrt{m}} e^{-z^2/2} = e^{-(x-z)^2/2}$$

[18] namely sequences $(a_m)_{m=1}^{+\infty}$ such that $\sum_{m=1}^{m} < +\infty$.
[19] Is ϕ well defined (i.e. $\phi(x) \in \ell^2$?). **Exercise!**

The Gaussian Kernel Feature Map (Continued)

We proved that the Gaussian kernel is indeed a kernel by **explicitly** finding $\mathcal{H}$ and a feature map ϕ.

This was fun, but doing it **for every kernel** gets old pretty quickly.

As already stressed many times, we **don't need** to know the feature map associated to a kernel. So...

Is there a way to show that a function is a kernel without going the "feature map way"?

Some Rules

Theorem. Let $k_1, k_2 : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be two positive definite kernels, then

- αk_1 is a kernel for any $\alpha \geq 0$,
- **Sum.** $k_1 + k_2$ is a kernel,
- **Point-wise product** $k(x, x') = k_1(x, x')k_2(x, x')$ is a kernel,
- **Composition** $k_1(\varphi(z), \varphi(z'))$ is a kernel for any $\varphi : \mathcal{Z} \to \mathcal{X}$

Let us look at a few examples...

Examples of Kernels

Polynomial Kernel. The function

$$k(x, x') = (x^\top x' + \alpha)^p$$

with $\alpha \geq 0$ is a kernel because:

- $k_1(x, x') = x^\top x'$ is a kernel,
- a constant function $k_2(x, x') = \alpha$ is a kernel (exercise?),
- $k_3 = k_1 + k_2$ is a kernel,
- $k = k_3^p$ is a power of a kernel (k_3 times itself p times).

Examples of Kernels: Polynomial of Kernels

Polynomials OF Kernels The function

$$k(x, x') = \sum_{m=1}^{p} \alpha_m \tilde{k}(x, x')^m$$

Is a kernel (sum of powers of kernels).

Series of Kernels. We could send $m \to +\infty$ in the above construction. If the series converges **then it is a kernel!**

For example, let $\alpha_m = 1/m!$ and $\tilde{k}(x, x') = (x^\top x')^m$ the polynomial kernel, we obtain

$$k(x, x') = \sum_{m=1}^{+\infty} \frac{(x^\top x')^m}{m!} = e^{xx'}$$

the exponential kernel.

The Gaussian Kernel (Again)

We can use the rules above to show that $k(x, x') = e^{-\|x-x'\|^2}$ is a kernel.

Expanding (again) the square we have

$$k(x, x') = e^{-(\|x\|^2/2 - xx' + \|x'\|^2/2)}$$

We can write the above equation as the point-wise product

$$k(x, x') = \underbrace{e^{-\|x\|^2} e^{-\|x'\|^2}}_{k_1(x,x')} \underbrace{e^{xx'}}_{k_2(x,x')}$$

Where,

- $k_1(x, x') = \phi_1(x)\phi(x')$ is a kernel with feature map $\phi(x) = e^{-\|x\|^2}$
- k_2 is the exponential kernel.

Hence k is a kernel.

Going Deeper with Kernels

We barely scratched the **surface** of what can be studied about kernels.

Given the scope of this module, we will limit ourselves to the **algorithmic perspective** on kernels and corresponding Hilbert spaces that we introduced in this class.

However, if you are interested in this area, I would strongly recommend **COMP0083 Advanced Topics in Machine Learning (ATML)**, which offers a nice complement to this module with topics on:

- Reproducing Kernel Hilbert Spaces
- Convex optimization

Part 6: Regularization

The Problem with Large Feature Spaces

Kernels allow to **efficiently** work with **large (or infinite!)** feature spaces!

However, there is a potential problem[20]...

The number n of training points is:

- **Finite** (can't really have an infinite dataset, can we?)
- Potentially **Smaller** than the feature space dimension D (Think about polynomial of Gaussian kernels).

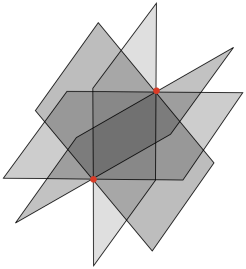
[20]isn't there always one?

Fitting Linear Models

A linear $f : \mathbb{R}^D \to \mathbb{R}$ is **uniquely** characterized by $n = D$ points.

(for f affine you need $n = D + 1$)

If $n < D$ you will have **more than one** possible functions.

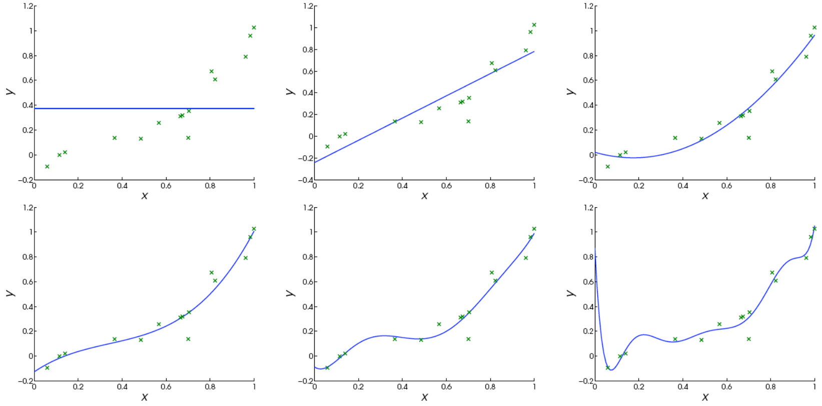


This is because $\hat{w} = (\Phi^\top \Phi)^{-1} \Phi^\top y$ corresponds to **solving** $\Phi \hat{w} = y$

Namely linear system of:

- n **equations** (one per point),
- D **variables**

Over Fitting your Points



Example: same number of points, polynomials of degree $p = 0$ to 5.

How Does Overfitting Arise?

Mechanically, we can see this issue arising in $\hat{w} = (\Phi^\top \Phi)^{-1} \Phi^\top y$ since

$$n < D \quad \Longrightarrow \quad \Phi^\top \Phi \quad \text{not invertible}$$

However, we saw that the lack of invertibility it is not a problem since:

- We can use the **dual** $\hat{w} = \Phi^\top (\Phi \Phi^\top)^{-1} y$, or better
- Use the **pseudoinverse**, rather than the inverse, of $\Phi \Phi^\top$.

However, the Pseudoinverse is not a panacea.

Instability of the Pseudoinverse on Small Principal Directions

Let $\Phi = U\Sigma V^\top$, with rank $r = n$ and smallest singular value $\sigma_n = 10^{-p}$.

We have shown that the squared loss minimizer is

$$\hat{w} = (\Phi^\top \Phi)^\dagger \Phi^\top y = V\Sigma^\dagger U^\top y$$

Let $x = v_n$ be the n-th column of V. Let $x' = (1+\epsilon)x$ and $\epsilon \in \mathbb{R}$. Then

$$|\hat{f}(x) - \hat{f}(x')| = |\epsilon| \; |v^\top V \Sigma^\dagger U^\top y| = \frac{|\epsilon|}{\sigma_n} |u_n^\top y|$$

Since $\sigma_n = 10^{-p}$, this implies that two points x and x' that are close ϵ to each other, will be predicted to have outputs that are

$$10^p \epsilon |u_n^\top y|$$

Namely, small "input" perturbations along v_n will be inflated by $\times 10^p$

Monkey-Patching the Pseudoinverse?

If the problem are small singular values, can we just get rid of them?

Idea. Choose a threshold $\lambda > 0$ and set to zero any $\Sigma_{ii} \leq \lambda$

Then, we would have the solution

$$\hat{w}_\lambda = V P_\lambda(\Sigma)^\dagger U^\top y$$

Where[21] $P_\lambda : \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n}$

$$\Big(P_\lambda(\Sigma)\Big)_{ii} = \begin{cases} \frac{1}{\Sigma_{ii}} & \text{if} \Sigma_i i > \lambda \\ 0 & \text{otherwise} \end{cases}$$

[21]More rigorously, P_λ sends diagonal matrices into diagonal matrices.

Regularization(s)

Yes we can! $\hat{w}_\lambda = V P_\lambda(\Sigma)^\dagger U^\top y$ is one example of **regularization**.

We have seen another example in the previous lecture:

$$\hat{w}_\lambda = (\Phi^\top \Phi + \lambda I)^{-1} \Phi^\top y$$

that has also an interpretation as the solution of

$$\hat{w}_\lambda = \underset{w \in \mathbb{R}^D}{\operatorname{argmin}} \ \frac{1}{n} \sum_{i=1}^{n} (w^\top \phi(x_i) - y_i)^2 + \lambda \|w\|^2$$

Also this solution "dampens" small singular values since

$$\hat{w}_\lambda = V(\Sigma^2 + \lambda I)^{-1} \Sigma U^\top y$$

Thus, the smallest singular value is always larger than $1/\lambda$.

Questions...

By modifying $\hat{w}$ into $\hat{w}_\lambda$ we are not fitting anymore our data...

$$\frac{1}{n}\sum_{i=1}^{n}(\hat{w}^\top\phi(x_i) - y_i)^2 \quad \leq \quad \frac{1}{n}\sum_{i=1}^{n}(\hat{w}_\lambda{}^\top\phi(x_i) - y_i)^2$$

Is this a problem? Maybe not.

Recall. we actually care about the expected risk

$$\mathcal{E}(\hat{w}_\lambda) = \mathbb{E}_{(x,y)}[(\hat{w}_\lambda^\top\phi(x) - y)^2]$$

Therefore (in principle at least) it might even happen that

$$\mathcal{E}(\hat{w}_\lambda) \quad \leq \quad \mathcal{E}(\hat{w})$$

Bias and Variance

Regularization has some **pros** and **cons**. **Recall:** let S a training set

$$\mathbb{E}_S[\mathcal{E}(\hat{w}_\lambda) - \mathcal{E}(f_*)] = Bias(\hat{w}_\lambda, f_*) + Variance(\hat{w}_\lambda)$$

- **Bias.** "Average" distance from the ground truth

$$Bias(\hat{w}_\lambda, f_*) = \mathbb{E}_x[(f_*(x) - \hat{w}_\lambda^\top \phi(x))^2]$$

  Hence, if $f_*(x) = \hat{w}^\top \phi(x)$, the larger λ the **further away** our predictions will be... **BAD!**

- **Variance.** Variance of models' output wrt training set S

$$Variance(\hat{w}_\lambda) = \mathbb{E}_{x,S}\left[|(\hat{w}_\lambda - \mathbb{E}_s[\hat{w}_\lambda])^\top \phi(x)|^2\right]$$

The larger λ the smaller the variance (**why?**)... **GOOD!**

Model Selection

Ok, so in theory we might have an advantage...

...but in practice what should we do?

Model Selection. Choose the best λ according $\hat{w}_\lambda$'s performance:

- **Hold-out.** Split training set $S = S_{tr} \cup S_{val}$
    - Choose a few candidate $\lambda_1, \dots, \lambda_m$
    - Train $\hat{w}_\lambda$ on S_{tr} for each λ_j
    - Evaluate each $\hat{w}_\lambda$ on S_{va}
    - Choose $\hat{w}_{\lambda_*}$ the best performing $\lambda_* = \lambda_{j_*}$
- **K-fold Cross-validation.** Consider K train-val splits of S.
    - Perform hold-out for each of the splits.
    - Return λ_* with the best w_{λ_*} on average.
- ...

Wrapping up

In this class:

- We have introduced feature maps to go beyond linear models,
- Leverage the Dual formulation when $n < D$
- Observed how inner products might be performed “implicitly”,
- Introduced rules to build kernels,
- Discusses how to regularize learning problems. $=$

Next class:

we will focus on classification problems and consider a method tailored for these setting: Support Vector Machines.