

Recap: kernel machines are an approach to ML
with nonlinear features

Hilbert space
↓ (vector space)

- Pick a nonlinear feature map $\phi: \mathbb{R}^d \rightarrow \mathcal{H}$

$$\text{ex: } \phi: [x_1, x_2] \rightarrow \{x_1, x_2, x_1^2, x_1 x_2, x_2^2\}$$

$$\phi: [x_1, x_2] \rightarrow \left[\sqrt{x_1^2 + x_2^2} \quad \tan^{-1}\left(\frac{x_2}{x_1}\right) \right]$$

$$\phi: [x_1, x_2] \rightarrow \left[\text{all the monomials in } x_1 \text{ and } x_2 \right]$$

- We showed if we want to learn a function of the form

$$f(x) = \langle w, \phi(x) \rangle$$

this can be accomplished using just the kernel function

$$K(x, y) = \langle \phi(x), \phi(y) \rangle$$

so learning can be done using K and the kernel matrix

$$K_{ij} = K(x_i, x_j)$$

In particular, the optimal f for an ERM takes the form

$$f(x) = \sum_{i=1}^n \alpha_i K(x, x_i)$$

so we just have to learn the α weights

Last example: logistic regression w/ non-linear features

$$\omega^* = \underset{\omega}{\operatorname{argmin}} \quad \frac{1}{n} \sum_{i=1}^n \log(1 + e^{-y_i \langle \omega, \phi(x_i) \rangle}) + \frac{\lambda \|\omega\|_2^2}{2}$$

we know in fact $\underline{\underline{f(x_i) = \langle \omega^*, \phi(x_i) \rangle}}$

$$= \sum_{j=1}^n \alpha_j^* K(x_i, x_j)$$
$$= \underline{\underline{[K\alpha]_i}} \leftarrow \begin{array}{l} \text{i-th row of} \\ K\alpha \end{array}$$

so we can predict using $f(x) = \sum_{i=1}^n \alpha_i K(x, x_i)$

if we find the optimal α :

$$\alpha_* = \underset{\alpha}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n \log(1 + e^{-y_i(K\alpha)_i}) + \frac{\lambda}{2} \alpha^T K \alpha$$

kernel logistic regression (recall $K \in \mathbb{R}^{n \times n}$)

Question: how can we gain accuracy like kernel methods while having a faster runtime?

An answer:

- note the cost of kernel methods is that of

1) multiplying by K $O(n^2)$

- for example in subgradient descent, or GD

2) inverting K $O(n^3)$

- for example in ridge regression

$$\alpha = (K + n\lambda I)^{-1}y$$

- approximate K by a low-rank matrix

$$K \approx CC^T \quad \text{where } C \in \mathbb{R}^{n \times r}$$

$$\begin{matrix} n & & r \\ \boxed{K} & \approx & \boxed{C} \end{matrix} \quad \begin{matrix} r & & n \\ & \boxed{C^T} & \end{matrix}$$

call $\tilde{K} = CC^T$ our low-rank approximation.
and replace K in all our algorithms with $\tilde{K}$.

Now

1) cost of multiplying by $\tilde{K}$ is $O(nr)$

consider for example $n = 50K$ and $r = 1000$

2) cost of inverting $\tilde{K} + \lambda n I$ $O(nr^2)$
(use the Woodbury Identity)

Two approaches to getting these low rank approximations:

1) Nyström approximations (originated in PDE literature)

2) Random feature maps (Recht & Rahimi, 2013)

Nyström approximations

Note that $K = K^T$ b/c $K(x_i, x_j) = K(x_j, x_i)$

write

$K =$

W	C^T
C	

where C are d sampled columns and W is the principal-submatrix where C and C^T overlap

note that this is equivalent to selecting l 'landmark points' $p_1, \dots, p_l$ and comparing all of our training points to these landmark points

$$C_{i,j} = K(x_i, p_j)$$

and now we extend these comparisons to the unobserved points (the non-landmark points) by noting that

$$K \approx C W^+ C^T$$

$$\begin{matrix} l & l & n \\ \boxed{C} & \boxed{W^+} & \boxed{C^T} \\ n & & \end{matrix}$$

where W^+ is the Moore-Penrose pseudoinverse of W . We call W^+ the coupling matrix

Why is $R \approx CW^+C^T$?

- Assume W is full rank, then $W^+ = W^{-1}$

and

$$CW^+C^T = \underbrace{\begin{bmatrix} W \\ L \end{bmatrix}}_C W^+ [W \mid L^T]$$

$$= \begin{bmatrix} W & L^T \\ L & LW^+L^T \end{bmatrix}$$

$$\text{so } K - CW^+C^T = \begin{bmatrix} 0 & 0 \\ 0 & \underbrace{\bar{K} - LW^+L^T}_{\text{Schur complement of } W \text{ in } K} \end{bmatrix}$$

$$\text{if } K = \begin{bmatrix} W & L^T \\ L & \bar{K} \end{bmatrix}$$

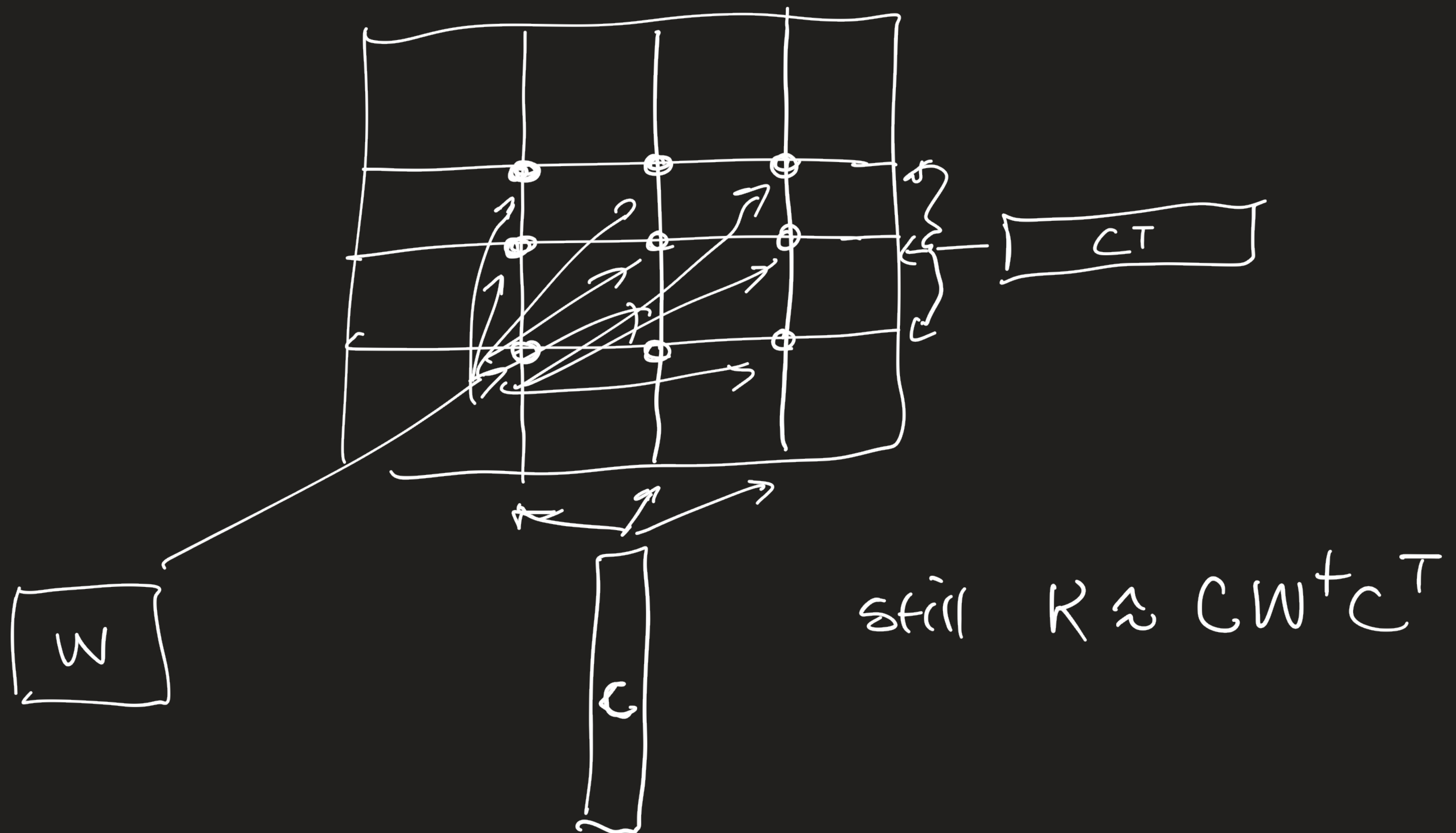
Schur complement of
 W in K

$$\text{Fact: } \text{rk}(W) = r(K) \text{ then } \bar{K} - LW^+L^T = 0$$

Take-away: if $\text{rk}(W) = \text{rk}(K)$ then

$$K = CW^+C^T$$

Note we could pick any $\underline{C}$ subset of columns of R



Nystrom alg

Input: K - kernel function

X - training data

l - # of landmark points

Output: C, W

use uniform sampling
w/ or w/o replacement

Alg

1. randomly pick l landmark points $\rightarrow p_1, \dots, p_l$

2. construct C by comparing all of the datapoints to my landmark points

$$C_{i,j} = K(x_i, p_j)$$

3. construct W by comparing the landmark points to themselves

$$W_{i,j} = K(p_i, p_j)$$

return C, W

Random feature maps

Motivation:

$$K = \Phi \Phi^T \quad \text{where} \quad \Phi = \begin{matrix} & \begin{matrix} D \end{matrix} \\ \begin{matrix} n \end{matrix} & \begin{bmatrix} \phi(x_1)^T \\ \vdots \\ \phi(x_n)^T \end{bmatrix} \end{matrix}$$

would be a low-rank approximation if

$$D \ll n$$

But in general $D \gg n$ (that's why we work with the kernel K instead of Φ directly)

What we want to do is find an approximate

feature map $\tilde{\phi} : \mathbb{R}^d \rightarrow \mathbb{R}^E$

where $\tilde{\Phi}$ is chosen so:

$$- E \ll n$$

$$- \langle \tilde{\Phi}(x), \tilde{\Phi}(y) \rangle \approx K(x, y)$$

because then

$$K_{ij} = K(x_i, x_j) \approx \langle \tilde{\Phi}(x_i), \tilde{\Phi}(x_j) \rangle$$

so

$$K \approx \tilde{\Phi} \tilde{\Phi}^T \quad \text{where} \quad \tilde{\Phi} = \begin{bmatrix} \tilde{\Phi}(x_1)^T \\ \vdots \\ \tilde{\Phi}(x_n)^T \end{bmatrix} \in \mathbb{R}^{n \times E}$$

and $E \ll n$ means this is a low-rank approx,
so is cheaper to use than K

Q: how to find these approx feature maps

A: randomness!

For a given kernel K , choose $\tilde{\phi}$ ^{← a random function} so that

$$\mathbb{E} \langle \tilde{\phi}(x), \tilde{\phi}(y) \rangle = K(x, y)$$

Ex: if $K(x, y) = e^{-\frac{\|x-y\|_2^2}{2\sigma^2}}$

then $\tilde{\phi}(x) = \sqrt{2} \cos(\omega^T x + b)$ where $\omega \sim \mathcal{N}(0, \sigma^2 I)$
and $b \text{ unif} \in [0, 2\pi]$

satisfies

$$\mathbb{E} \langle \tilde{\phi}(x), \tilde{\phi}(y) \rangle = K(x, y)$$

$$K \in \mathbb{R}^{n \times n}$$

costs $\mathcal{O}(n^2)$ mult

$\mathcal{O}(n^3)$ invert

$$K \approx \tilde{\Phi} \tilde{\Phi}^T \text{ where } \tilde{\Phi} \in \mathbb{R}^{n \times E}$$



randomized, low-dim (E)

approx feature map

costs $\mathcal{O}(nE)$ mult

costs $\mathcal{O}(nE^2)$ invert



$$\Phi \Phi^T \text{ where } \Phi \in \mathbb{R}^{n \times D} \text{ where } D > n$$

deterministic, high-dim (D)

exact feature map

costs $\mathcal{O}(nD) = \mathcal{O}(n^2)$

cost $\mathcal{O}(nD^2) = \mathcal{O}(n^3)$