CS 189/289

Today's lecture:

1. Kernel methods*.

Assigned reading: none (not in textbook)

**different from CNN kernels, and diffusion kernels*

Recall linear regression:

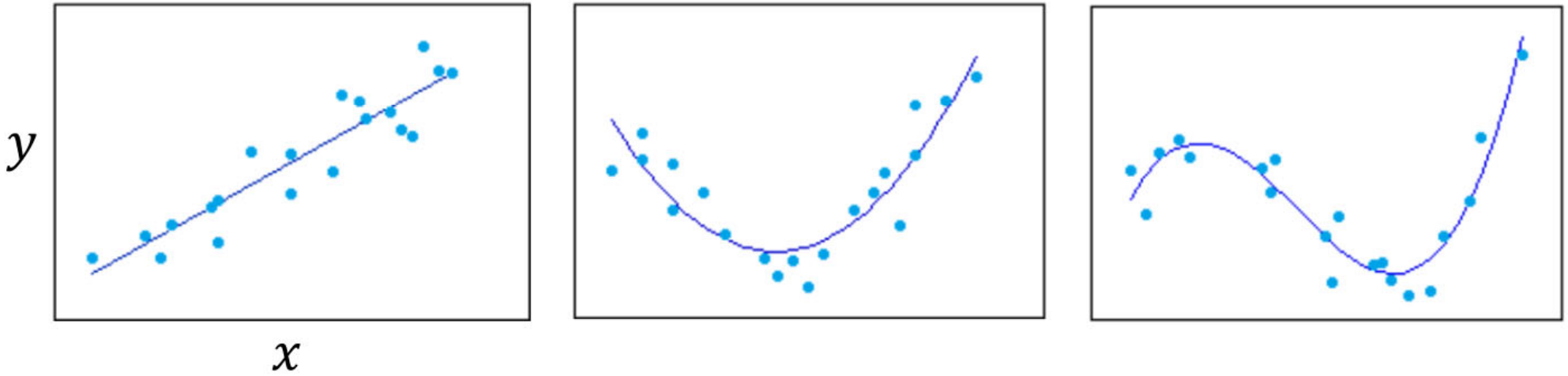
How useful can a linear model be?!

$w, x \in \mathbb{R}^1$

$\hat{y} = w^T x$ **Linear**

$\hat{y} = w^T [x, x^2]$ **Quadratic**

$\hat{y} = w^T [x, x^2, x^3]$ **Cubic**

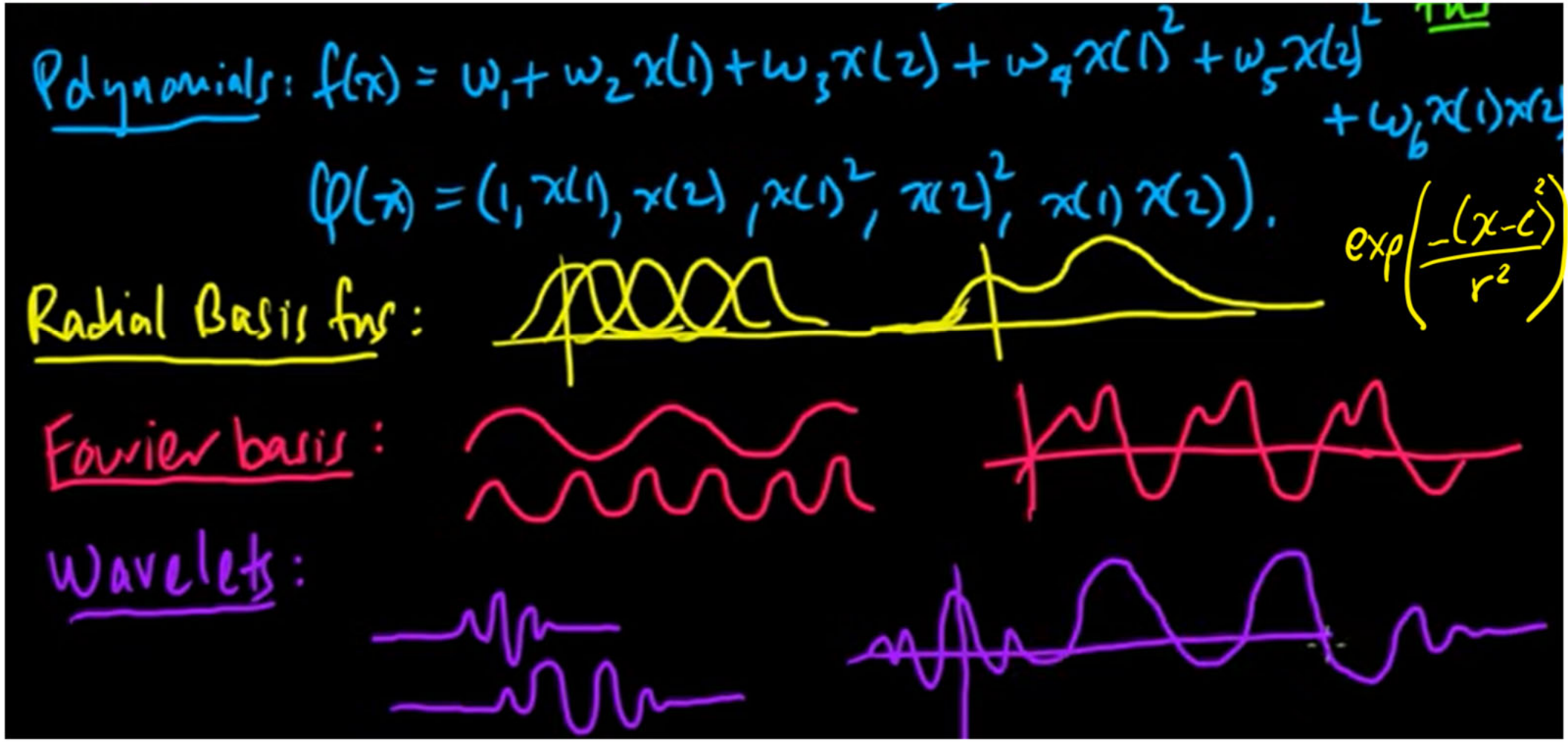


For full generality, $x \in \mathbb{R}^D$ need the cross-terms and bias terms for arbitrary polynomial, e.g., quadratic $[1, x_1, x_2, x_1^2, x_2^2, x_1 x_2]$.

https://statisticsbyjim.com/regression/curve-fitting-linear-nonlinear-regression/

Recall linear regression:

Many basis possible functions!



Recall linear regression:

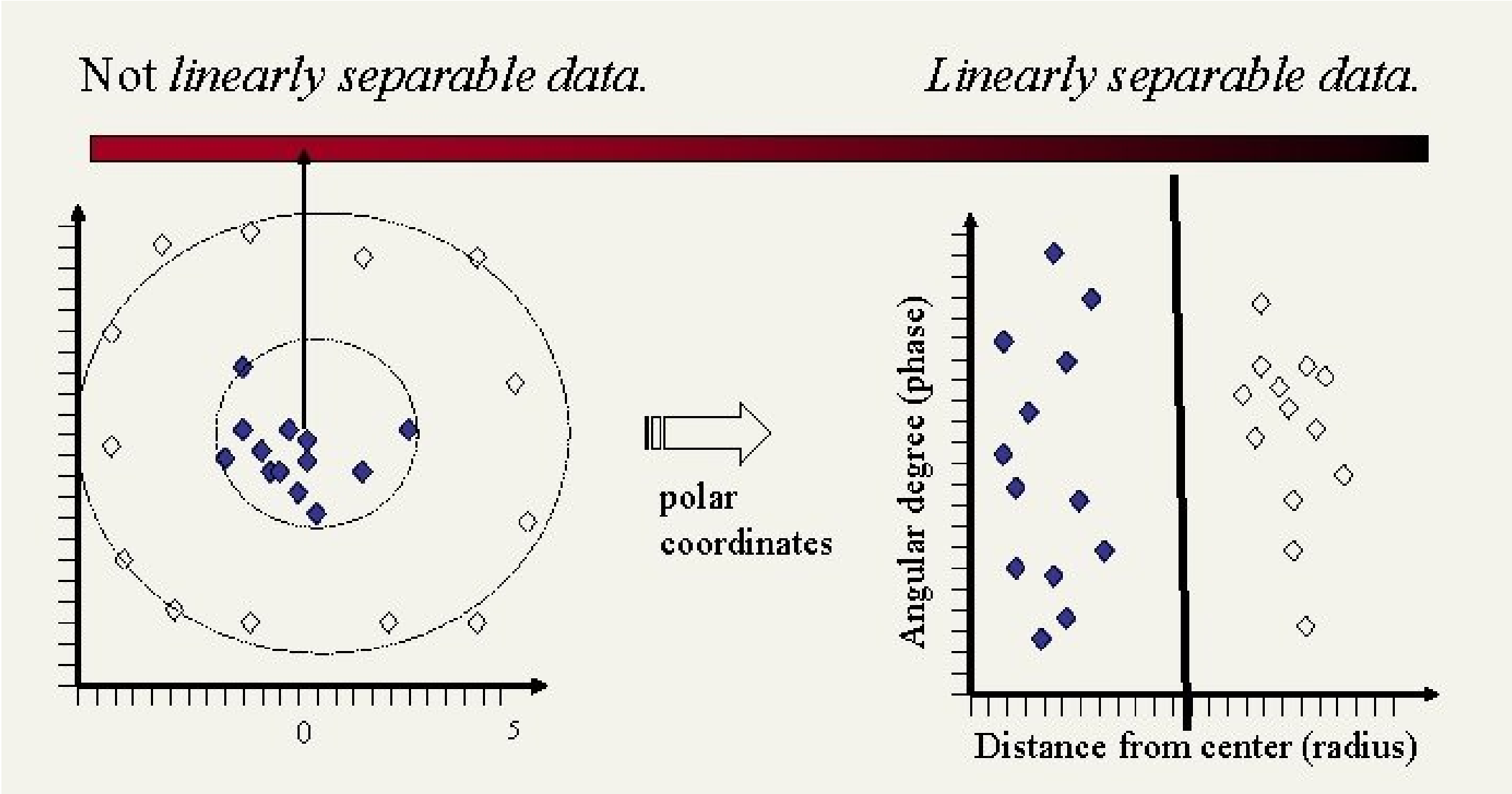
Basis expansion of raw input space

Because these basis functions are pre-determined before we do the regression, everything works out exactly the same, only the notation changes:

$$\hat{y} = E_Y[p(y|x)] = w^T \Phi(x), \text{ for } w \in \mathbb{R}^k, x \in \mathbb{R}^d$$

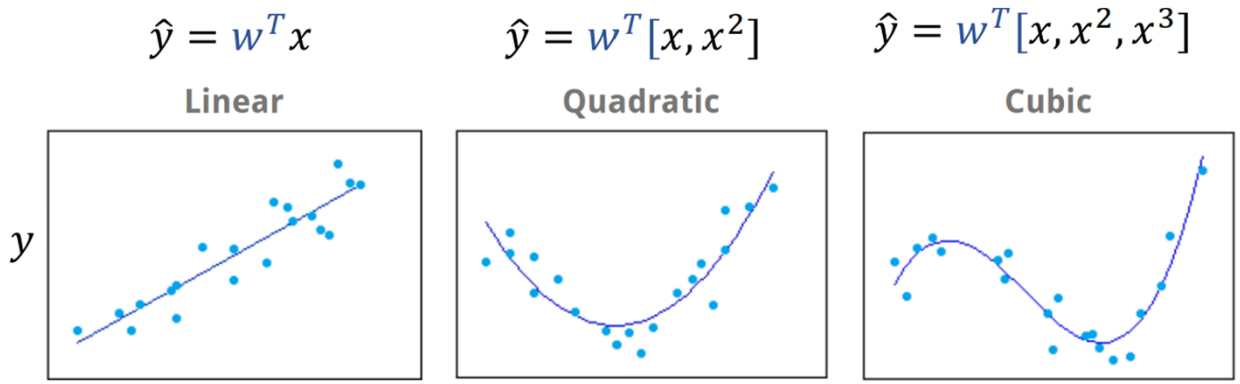
In this lecture, for simplicity of notation, we will assume that this expansion has already been done, and just write $\hat{y} = w^T x$.

May not even need *more* features



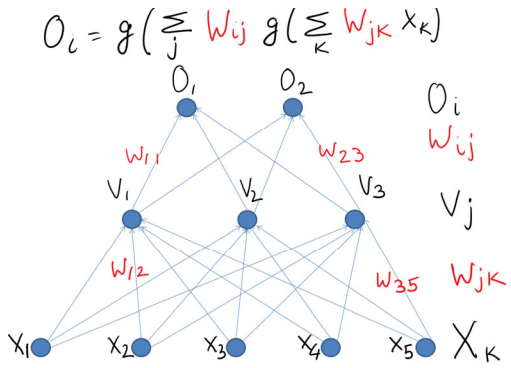
Some properties of linear/logistic regression

1. Given that we can expand using a complete basis, can capture *any* function (expansion may be arbitrarily large).
2. Knowing a suitable basis lets us truncate to use fewer basis functions.
3. Given a basis, objectives are convex, so optimization is easy (although costly if have many features).



Some properties of neural networks

1. *Adaptively learns "bases"* via the hidden layers.
2. Given that we can choose any architecture, we can choose any basis, and thus can capture *any* function.
3. But neural networks are highly non-convex (though practically we don't need/want global optimum).
4. And, need lots of data to learn good bases.



Back to linear regression: issue with basis expansion

Even if we know a good basis expansion, there may be too many basis vectors to deal with (computationally). *e.g.* polynomial basis:

d \ p (degree of polynomial basis; dimensions of raw input, $\vec{x}$)	0	1	2	3	$\cdots$
1 (univariate)	1	x	x^2	x^3	$\cdots$
2 (bivariate)	1	x_1, x_2	x_1^2, x_2^2, x_1x_2	$x_1^3, x_2^3, x_1^2x_2, x_1x_2^2$	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

d \ p	1	3	5	10	25
1	2	4	6	11	26
3	4	20	56	286	3276
5	6	56	252	3003	142506
10	11	286	3003	184756	183579396
25	26	3276	142506	183579396	126410606437752

polynomial degree p for $x \in \mathbb{R}^d$

#features=$\binom{d+p}{d} = \frac{(d+p)!}{d!p!}$

https://www.eecs189.org/static/notes/n3.pdf

To the rescue: the "kernel trick"

- The kernel provides a way to compute the dot product between two vectors, X and Y, in some high-dimensional space <u>without projecting the vectors to that space</u>: $K(x_i, x_j) = K_{i,j} = \Phi(x_i^T)\Phi(x_j)$.
- If we can massage our loss functions and prediction functions to use only dot products between feature vectors, $x_i^T x_j$, then we can use only kernel computations, $K_{i,j} = \Phi(x_i^T)\Phi(x_j) \in \mathbb{R}^1$, never using $\Phi(x) \in \mathbb{R}^{d+p\ Choose\ d}$.
- $K_{i,j} = \Phi(x_i^T)\Phi(x_j)$--called the *kernel function*.
- Then can work only with the pairwise entries in $K \in \mathbb{R}^{n \times n}$, *i.e.* dependencies on n, not d!

Lets see how this kernel trick works:

Consider a kernel function with $x, z \in \mathbb{R}^{d=2}$:

$$K(x,z) = \left(x^T z + 1\right)^2 \quad = (x^T z + 1)^2$$

$$= (x_1 z_1 + x_2 z_2 + 1)^2$$

$$= x_1^2 z_1^2 + x_2^2 z_2^2 + 1 + 2x_1 x_2 z_1 z_2 + 2x_1 z_1 + 2x_2 z_2$$

$$= (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2, x_1^2, x_2^2) \cdot (1, \sqrt{2}z_1, \sqrt{2}z_2, \sqrt{2}z_1 z_2, z_1^2, z_2^2)$$

$$= \Phi^T(x)\Phi(z)$$

for $\phi(x) = (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2, x_1^2, x_2^2)$.

- Contains all the terms from a $p = 2$ polynomial expansion: $1, x_1, x_2, x_1 x_2, x_1^2, x_2^2$.
- Instead of explicitly computing $\Phi(x) \in \mathbb{R}^{d'=6}$ and $\Phi(z) \in \mathbb{R}^{d'=6}$ to compute $\Phi^T(x)\Phi(z)$, just compute $x^T z + 1 \in \mathbb{R}$ and then square it!

Kernels for polynomial basis expansion

It turns out that in general, computing

$$K(x, v) = \Phi(x) \cdot \Phi(v) = (1 + x \cdot v)^p$$

corresponds to a polynomial basis expansion of order $\boldsymbol{p}$.

This is called the *polynomial* kernel function, of which specific examples include:

$$\text{quadratic} \quad K^{\text{quad}}(\mathbf{x}, \mathbf{v}) = (1 + \mathbf{x} \cdot \mathbf{v})^2$$

$$\text{cubic} \quad K^{\text{cubic}}(\mathbf{x}, \mathbf{v}) = (1 + \mathbf{x} \cdot \mathbf{v})^3$$

Why kernels methods are important today

- Historically, some supervised kernel methods were state-of-the-art (*e.g.*, SVMs); still useful, but fallen by the wayside.
- Can make many useful, simple algorithms have richer capacity (*e.g.*, PCA, clustering, linear/logistic regression)--next.
- Can tailor kernels to specific kinds of objects, such as graphs.
- Methods for which our *uncertainty estimation* is often best are based on kernels (Gaussian Process regression).
- Useful for density estimation.
- Concept infiltrates much modern day research.
- Forms basis of *neural tangent kernels*, which drives some theoretical ML research of understanding of deep neural networks.

What if we wanted to do PCA in polynomial basis degree p?

Can we do PCA if don't have X explicitly?

- Suppose you're given pairwise distances between n cities, $M \in \mathbb{R}^{n \times n}$, and asked you to find a 2D representation?
- Think of $M = XX^T \in \mathbb{R}^{n \times n}$ for some <u>unobserved X</u> (instead of $X^T X$ as with PCA).
- Now new basis is in U from: $M = USV^T$.

We just performed *Multidimensional Scaling* (MDS):

- Implicitly assumes some latent space X of unknown dimension for which the distance is an inner product distance, $\mathrm{d}(\mathrm{x}', \mathrm{x}) = x' x^T$.
- Could be non-linear distance function of actual x (*e.g.*, if latent space had a polynomial expansion).

- Example: given pairwise distances between cities

	Atl	Chi	Den	Hou	LA	Mia	NYC	SF	Sea	DC
Atlanta	0									
Chicago	587	0								
Denver	1212	920	0							
Houston	701	940	879	0						
LA	1936	1745	831	1374	0					
Miami	604	1188	1726	968	2339	0				
NYC	748	713	1631	1420	2451	1092	0			
SF	2139	1858	949	1645	347	2594	2571	0		
Seattle	2182	1737	1021	1891	959	2734	2406	678	0	
DC	543	597	1494	1220	2300	923	205	2442	2329	0

- We showed that MDS was the same as PCA, except we started with the $n \times n$ *gram matrix*, $M = XX^T$.
- We can use a kernel to compute

$K_{i,j} = M = K = \Phi^T(x_i)\Phi(x_j)$.

- This is *kernel PCA*!

What constitutes a valid kernel function?

- So far we saw/proved the polynomial kernel of degree p,
$$K(x, v) = \Phi(x) \cdot \Phi(v) = (1 + x \cdot v)^p.$$
- Suppose someone gives us only a function, $M(x, v)$—how can we know if it corresponds to a valid kernel function?
- That is how can we know if there exists some mapping $\mathrm{x} \to \Phi(x)$ such that $M(x, v) = \Phi(x) \cdot \Phi(v) = K(x, v)$?

<u>Mercer's Theorem</u>: iff for any finite set of data $\{x_i\}_{i=1}^n$, the matrix $M_{i,j} = \{M(x_i, x_j)\}$ is positive semi-definite (PSD), then $\exists\ \Phi(x)$ such that $M(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j)$, and $M(x, v)$ is a kernel function, $K(x, v)$.

The art of constructing a kernel method

1. Take a known non-kernel algorithm (PCA, linear regression, logistic regression, k-means, *etc.*), and re-write the training (and testing if appropriate) algorithm in terms of only inner products, $\{x_i^T x_j\}_{i,j}$.
2. Replace all occurrences of $x_i^T x_j$ with $K(x_i, x_j)$ (never compute $\Phi(x)$—you may know even know it, or it might be infinite dimensional!)
3. Proceed as you would otherwise (minimize the loss, *etc.*).

Combining kernels to get a new kernel

- Suppose we have two kernels

$$k_1 : \mathcal{X} \times \mathcal{X} \to \mathbb{R} \qquad k_2 : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$$

  defined on data space *X*

- Then the following functions are valid kernels:

$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}')$$

$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}')\, k_2(\mathbf{x}, \mathbf{x}')$$

$$k(\mathbf{x}, \mathbf{x}') = c\, k_1(\mathbf{x}, \mathbf{x}') \text{ for } c > 0$$

$$k(\mathbf{x}, \mathbf{x}') = f(k_1(\mathbf{x}, \mathbf{x}'))$$

  where *f* is a polynomial with positive coefficients or the exponential function

(slide from Andreas Krause)

Examples of valid kernels

- Polynomial (on $\mathbb{R}^d$):

$$K(x, x') = (x.x' + 1)^d$$

(slide from Jean-Phillipe Vert)

Examples of valid kernels

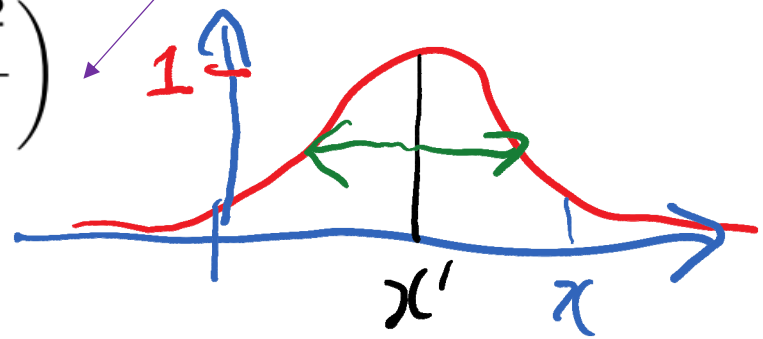
- Polynomial (on $\mathbb{R}^d$):

$$K(x, x') = (x.x' + 1)^d$$

- Gaussian radial basis function (RBF) (on $\mathbb{R}^d$)

$$K(x, x') = \exp\left(-\frac{||x - x'||^2}{2\sigma^2}\right)$$

can be shown that this corresponds to infinite-dimensional $\Phi(x)$!



(slide from Jean-Phillipe Vert)

Examples of valid kernels

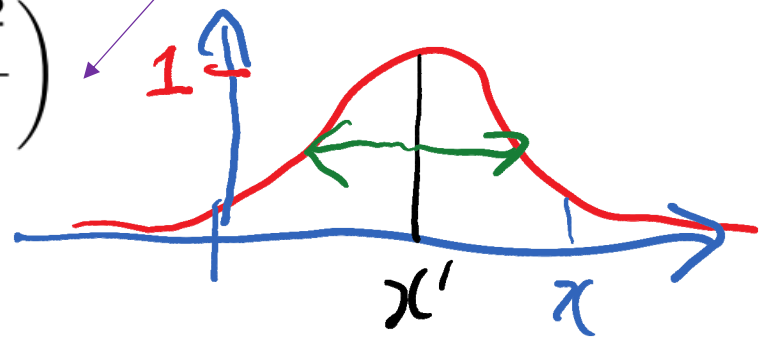
- Polynomial (on $\mathbb{R}^d$):

$$K(x, x') = (x.x' + 1)^d$$

- Gaussian radial basis function (RBF) (on $\mathbb{R}^d$)

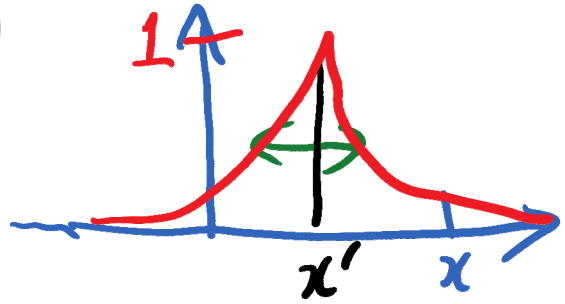
$$K(x, x') = \exp\left(-\frac{||x - x'||^2}{2\sigma^2}\right)$$

can be shown that this corresponds to infinite-dimensional $\Phi(x)$!



- Laplace kernel (on $\mathbb{R}$)

$$K(x, x') = \exp\left(-\gamma|x - x'|\right)$$



(slide from Jean-Phillipe Vert)

Examples of valid kernels

(more technical details on kernels in JPV's slides here if interested).

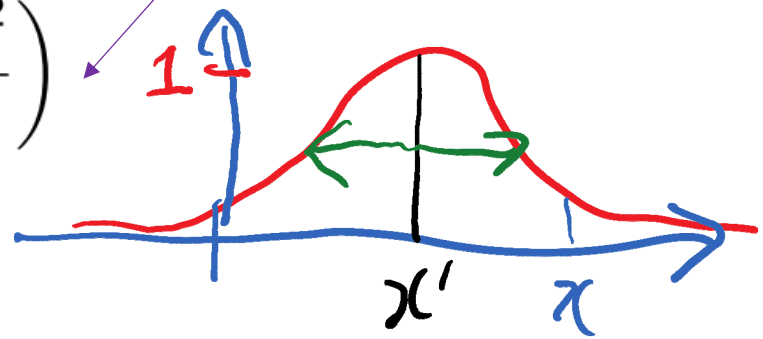
- Polynomial (on $\mathbb{R}^d$):

$$K(x, x') = (x.x' + 1)^d$$

- Gaussian radial basis function (RBF) (on $\mathbb{R}^d$)

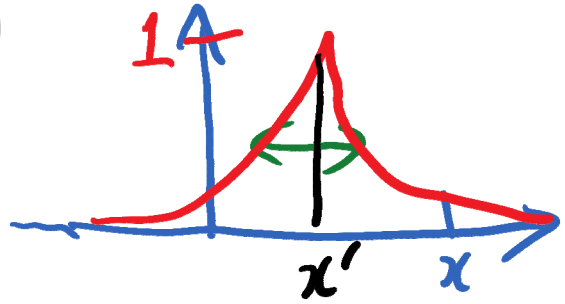
can be shown that this corresponds to infinite-dimensional $\Phi(x)$!

$$K(x, x') = \exp\left(-\frac{||x - x'||^2}{2\sigma^2}\right)$$



- Laplace kernel (on $\mathbb{R}$)

$$K(x, x') = \exp\left(-\gamma|x - x'|\right)$$



- Min kernel (on $\mathbb{R}_+$)

$$K(x, x') = \min(x, x')$$

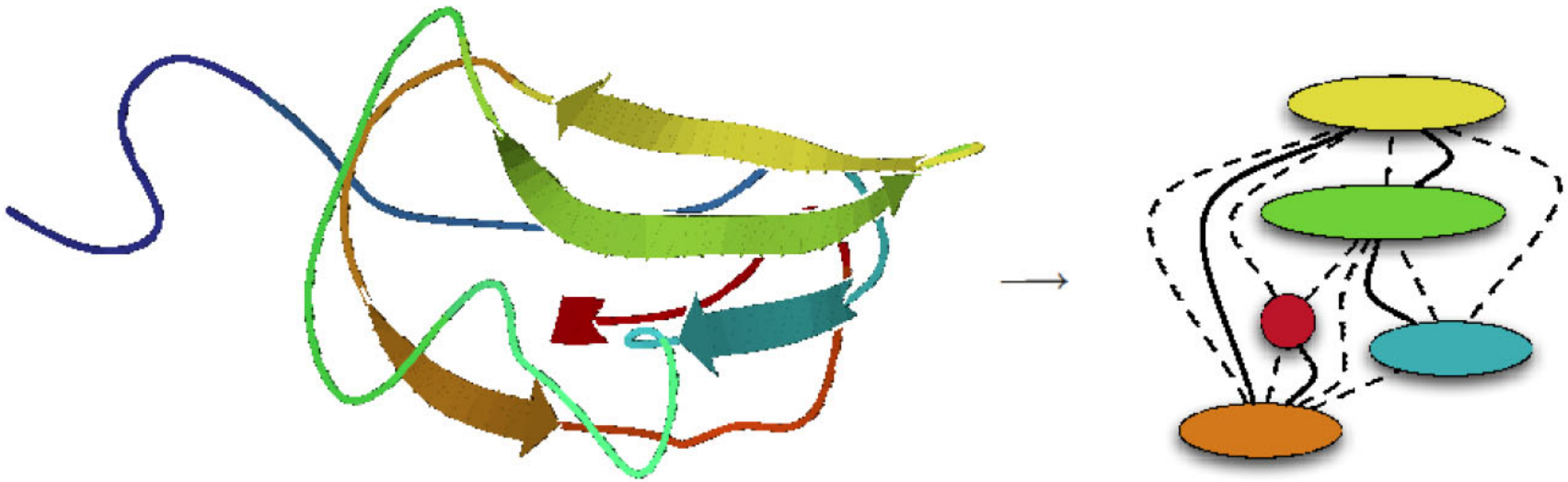
(slide from Jean-Phillipe Vert)

Example of non-valid kernel

- $K(x, x') = \sin(x)\cos(x')$—why is this not valid?
  It's not symmetric, $K(x, x') \neq K(x', x)$, so cannot be PSD.
- $K(x, x') = x^T M x'$—is this valid?
  It *might* be valid, but is not guaranteed:
  - Suppose $x \in \mathbb{R}^1$ and $M = -1$, then $K(x, x) = -x^T x$, which by itself is already not PSD.
  - Suppose M is PSD, then $M = UQ^{\frac{1}{2}}Q^{\frac{1}{2}}\mathrm{U}^{\mathrm{T}} \equiv H^T H$ thus $x^T M x' = (x^T H^T)(Hx') = \Phi^{\mathrm{T}}(x)\Phi(x')$, for $\Phi(x) = Hx$ which is a valid kernel!

Can have kernels on different objects!

e.g. kernels on graphs (compares two graphs)



[Borgwardt et al.]

- Can define a kernel for measuring similarity between graphs by comparing random walks on both graphs (not further defined here)

(slide from Andreas Krause)

e.g. Kernelized ridge regression

- Lets work with the "squared error" version of the loss function:

$$L\left(w \in \mathbb{R}^d\right) = \frac{1}{2}\sum_i \left(y_i - w^T x_i\right)^2 + \lambda w^T w \qquad (1)$$

e.g. Kernelized ridge regression

- Lets work with the "squared error" version of the loss function:

$$L(w \in \mathbb{R}^d) = \frac{1}{2}\sum_i (y_i - w^T x_i)^2 + \lambda w^T w \qquad (1)$$

- We will show that the solution to this (the MLE), $\hat{w}$, can be written as a linear combination of the training data, $\hat{w} = \alpha^T X$:

$$\nabla_{\mathrm{w}^T} L(w) = \sum_i -(y_i - w^T x_i)x_i + \lambda w = 0$$

$$\Rightarrow \hat{w}^T = 1/\lambda \sum_{i=1}^{n} (y_i - w^T x_i)x_i = \sum_{i=1}^{n} \alpha_i x_i = \alpha^T X \in \mathbb{R}^{1\times d} \qquad (\text{for } y, \alpha \in \mathbb{R}^{n\times 1}, X \in \mathbb{R}^{n\times d})$$

e.g. Kernelized ridge regression

- Lets work with the "squared error" version of the loss function:

$$L(w \in \mathbb{R}^d) = \frac{1}{2}\sum_i (y_i - w^T x_i)^2 + \lambda w^T w \qquad (1)$$

- We will show that the solution to this (the MLE), $\hat{w}$, can be written as a linear combination of the training data, $\hat{w} = \alpha^T X$:

$$\nabla_{\mathrm{w}^T} L(w) = \sum_i -(y_i - w^T x_i)x_i + \lambda w = 0$$

$$\Rightarrow \hat{w}^T = 1/\lambda \sum_{i=1}^{n} (y_i - w^T x_i)x_i = \sum_{i=1}^{n} \alpha_i x_i = \alpha^T X \in \mathbb{R}^{1\times d} \qquad (\text{for } y, \alpha \in \mathbb{R}^{n\times 1}, X \in \mathbb{R}^{n\times d})$$

Plug back into 1: $2 \times L(\alpha \in \mathbb{R}^n) = \cancel{2 \times \frac{1}{2}}(y - Xw)^T(y - Xw) + 2\lambda w^T w$

e.g. Kernelized ridge regression

- Lets work with the "squared error" version of the loss function:

$$L(w \in \mathbb{R}^d) = \frac{1}{2}\sum_i (y_i - w^T x_i)^2 + \lambda w^T w \qquad (1)$$

- We will show that the solution to this (the MLE), $\hat{w}$, can be written as a linear combination of the training data, $\hat{w} = \alpha^T X$:

$$\nabla_{\mathrm{w}^T} L(w) = \sum_i -(y_i - w^T x_i)x_i + \lambda w = 0$$

$$\Rightarrow \hat{w}^T = 1/\lambda \sum_{i=1}^{n} (y_i - w^T x_i)x_i = \sum_{i=1}^{n} \alpha_i x_i = \alpha^T X \in \mathbb{R}^{1\times d} \qquad (\text{for } y, \alpha \in \mathbb{R}^{n\times 1}, X \in \mathbb{R}^{n\times d})$$

Plug back into 1: $2 \times L(\alpha \in \mathbb{R}^n) = \cancel{2 \times \frac{1}{2}}(y - Xw)^T(y - Xw) + 2\lambda w^T w$

$$= y^T y - 2w^T X^T y + w^T X^T X w + 2\lambda w^T w$$

e.g. Kernelized ridge regression

- Lets work with the "squared error" version of the loss function:

$$L(w \in \mathbb{R}^d) = \frac{1}{2}\sum_i (y_i - w^T x_i)^2 + \lambda w^T w \qquad (1)$$

- We will show that the solution to this (the MLE), $\hat{w}$, can be written as a linear combination of the training data, $\hat{w} = \alpha^T X$:

$$\nabla_{\mathrm{w}^T} L(w) = \sum_i -(y_i - w^T x_i)x_i + \lambda w = 0$$

$$\Rightarrow \hat{w}^T = 1/\lambda \sum_{i=1}^{n} (y_i - w^T x_i)x_i = \sum_{i=1}^{n} \alpha_i x_i = \alpha^T X \in \mathbb{R}^{1\times d} \qquad (\text{for } y, \alpha \in \mathbb{R}^{n\times 1}, X \in \mathbb{R}^{n\times d})$$

Plug back into 1: $2 \times L(\alpha \in \mathbb{R}^n) = \cancel{2 \times \frac{1}{2}}(y - Xw)^T(y - Xw) + 2\lambda w^T w$

$$= y^T y - 2w^T X^T y + w^T X^T X w + 2\lambda w^T w$$

$$= y^T y - 2\alpha^T X X^T y + \alpha^T X X^T X X^T \alpha + \lambda \alpha^T X X^T \alpha$$

e.g. Kernelized ridge regression

- Lets work with the "squared error" version of the loss function:

$$L(w \in \mathbb{R}^d) = \frac{1}{2}\sum_i (y_i - w^T x_i)^2 + \lambda w^T w \qquad (1)$$

- We will show that the solution to this (the MLE), $\hat{w}$, can be written as a linear combination of the training data, $\hat{w} = \alpha^T X$:

$$\nabla_{\mathrm{w}^T} L(w) = \sum_i -(y_i - w^T x_i)x_i + \lambda w = 0$$

$$\Rightarrow \hat{w}^T = 1/\lambda \sum_{i=1}^{n} (y_i - w^T x_i)x_i = \sum_{i=1}^{n} \alpha_i x_i = \alpha^T X \in \mathbb{R}^{1\times d} \qquad (\text{for } y, \alpha \in \mathbb{R}^{n\times 1}, X \in \mathbb{R}^{n\times d})$$

Plug back into 1: $2 \times L(\alpha \in \mathbb{R}^n) = \cancel{2 \times \frac{1}{2}}(y - Xw)^T(y - Xw) + 2\lambda w^T w$

$$= y^T y - 2w^T X^T y + w^T X^T X w + 2\lambda w^T w$$

$$= y^T y - 2\alpha^T X X^T y + \alpha^T X X^T X X^T \alpha + \lambda \alpha^T X X^T \alpha$$

$$= y^T y - 2\alpha^T K y + \alpha^T K K \alpha + \lambda \alpha^T K \alpha, \text{ for } K = XX^T$$

e.g. Kernelized ridge regression

- Lets work with the "squared error" version of the loss function:

$$L(w \in \mathbb{R}^d) = \frac{1}{2}\sum_i (y_i - w^T x_i)^2 + \lambda w^T w \qquad (1)$$

- We will show that the solution to this (the MLE), $\hat{w}$, can be written as a linear combination of the training data, $\hat{w} = \alpha^T X$:

$$\nabla_{\mathrm{w}^T} L(w) = \sum_i -(y_i - w^T x_i)x_i + \lambda w = 0$$

$$\Rightarrow \hat{w}^T = 1/\lambda \sum_{i=1}^{n} (y_i - w^T x_i)x_i = \sum_{i=1}^{n} \alpha_i x_i = \alpha^T X \in \mathbb{R}^{1\times d} \qquad (\text{for } y, \alpha \in \mathbb{R}^{n\times 1}, X \in \mathbb{R}^{n\times d})$$

Plug back into 1: $2 \times L(\alpha \in \mathbb{R}^n) = \cancel{2 \times \frac{1}{2}}(y - Xw)^T(y - Xw) + 2\lambda w^T w$

$$= y^T y - 2w^T X^T y + w^T X^T X w + 2\lambda w^T w$$

$$= y^T y - 2\alpha^T X X^T y + \alpha^T X X^T X X^T \alpha + \lambda \alpha^T X X^T \alpha$$

$$= y^T y - 2\alpha^T K y + \alpha^T K K \alpha + \lambda \alpha^T K \alpha, \text{ for } K = XX^T = \Phi(X)\Phi(X^T)$$

e.g. Kernelized ridge regression

for $K = \Phi(X)\Phi(X^T)$

Setting the gradient to zero remains analytical:

$$\nabla_{\alpha^T} L(\alpha) = \nabla_\alpha \left(y^T y - 2\alpha^T K y + \alpha^T K K \alpha + \lambda \alpha^T K \alpha\right)$$

$$\begin{aligned}
\Rightarrow \quad 0 &= -2Ky + 2KK\alpha + 2\lambda K\alpha \\
\Rightarrow \quad 0 &= -Ky + (KK + \lambda K)\alpha \\
\Rightarrow \quad 0 &= -K^{-1}Ky + (K^{-1}KK + \lambda K^{-1}K)\alpha \\
\Rightarrow \quad y &= (K + \lambda I)\alpha \\
\Rightarrow \quad \hat{\alpha} &= (K + \lambda I)^{-1} y
\end{aligned}$$

- Reminiscent of standard ridge regression: $\hat{w} = \left(X^T X + \lambda I\right)^{-1} X y$
- Now we invert an $n \times n$ matrix, instead of $d \times d$!
- For large basis expansion, $n \ll d$.

e.g. Kernelized ridge regression

$$\hat{w} = \Phi(X^T)\alpha \in \mathbb{R}^d$$
$$\hat{\alpha} = (K + \lambda I)^{-1} y \in \mathbb{R}^n$$

Finally, to make a point prediction on test point, x_*, after having trained the model on training data, $X \in \mathbb{R}^{n \times d}, y \in \mathbb{R}^n$

$$\begin{aligned} y_* &= \Phi\left(x_*^T\right) w \\ &= \Phi\left(x_*^T\right) \Phi(X^T)\alpha \\ &= K(x_*, X)\alpha\,, \end{aligned}$$

where $K(x_*, X) \in \mathbb{R}^{1 \times n}$ and $K = \Phi(X)\Phi(X^T)\mathbb{R}^{\mathrm{n} \times n}$

$$= K(x_*, X)\,(K + \lambda I)^{-1} y$$

If we use an RBF kernel, then we will have done ridge regression with an infinite feature space, $\Phi\left(x_*^T\right) \in \mathbb{R}^\infty$!

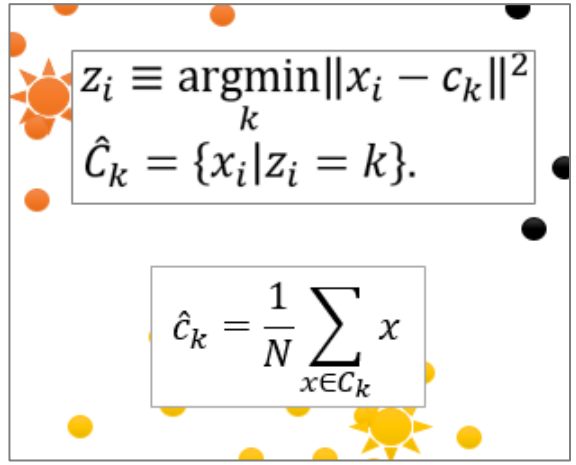
e.g. Kernelized k-means clustering

- Re-write Euclidean distance between two points in the augmented feature space in terms of kernel function:

$$\|\Phi(x_i) - \Phi(x_j)\|_2^2 = \Phi^T(x_i)\Phi(x_i) + \Phi^T(x_j)\Phi(x_j) - 2\Phi^T(x_i)\Phi(x_j)$$
$$= K(x_i, x_i) + K(x_j, x_j) - 2K(x_i, x_j)$$

- Write cluster centroids as a linear combination of basis-expanded training data points:

$$c_k = \sum_{i=1}^{n} \alpha_{ik}\Phi(x_i)$$



- Now, Euclidean distance between a point, $\boldsymbol{x}$, and a cluster center, in the augmented feature space is:

$$\|\Phi(x) - c_k\|_2^2 = K(x, x) + \sum_{i,j=1}^{n} \alpha_{ik}\alpha_{jk}K(x_i, x_j) - 2\sum_{i=1}^{n} \alpha_{ik}K(x, x_i) \boxed{\equiv d_k\ (x|\alpha)}$$

e.g. Kernelized k-means clustering

$$c_k = \sum_{i=1}^{n} \alpha_{ik} \Phi(x_i) \quad \alpha \in \mathbb{R}^{n \times k}$$

Re-write the original k-means loss function for $\{c_k\}$,

$$\hat{c}_k = \underset{c_k}{\operatorname{argmin}} \sum_{x \in C_k} \|x - c_k\|^2$$

in terms of this distance, $d_k(x|\alpha)$, to get a kernalized k-means loss for the cluster centers (conditioned on partition):

$$\hat{\alpha}_k = \arg\min_{\alpha} \sum_{\{x \in C_k\}} d_k(x_i|\alpha)$$

which gives us the cluster centers, from which we can iterate as in k-means by using $d_k(x_i|\alpha)$ to find which point belongs to which cluster.

Kernelized Mixture of Gaussians

This is much harder to derive.

International Conference on Algorithmic Learning Theory

ALT 2003: Algorithmic Learning Theory pp 159-174 | Cite as

Download book PDF

Kernel Trick Embedded Gaussian Mixture Model

Authors | Authors and affiliations

Jingdong Wang, Jianguo Lee, Changshui Zhang

Conference paper

14 Citations

439 Downloads

Dual Representations $\mathbb{R}^{\boldsymbol{n}}$ vs $\mathbb{R}^{\boldsymbol{d}}$

For k-means clustering, and linear regression, the kernalization step involved:

1. Noticing that the parameter solution in the non-kernalized version could be written as a linear combination of the inputs, $\boldsymbol{X}$, with coefficients $\boldsymbol{\alpha} = \{\alpha_1 \dots \alpha_n\}$ for $\boldsymbol{n}$ training data points.
2. Changing the loss function to be a function of $\boldsymbol{\alpha} \in \mathbb{R}^{\boldsymbol{n}}$ instead of the original parameters $\in \mathbb{R}^{\boldsymbol{d}}$.
3. Solving for the parameters in this *dual representation*, $\boldsymbol{\alpha}$.

<u>Fundamentally, the duality lets us work in $\boldsymbol{n}$ dimensions (# of training data points) instead of $\boldsymbol{d}$ (number of features space).</u>

Extras (not responsible for)

RBF corresponds to infinite dimensional $\Phi(x)$

Without loss of generality, let $\gamma = \frac{1}{2}$.

$$K_{\text{RBF}}(\mathbf{x}, \mathbf{x}') = \exp\left[-\frac{1}{2}\|\mathbf{x} - \mathbf{x}'\|^2\right]$$

$$= \exp\left[-\frac{1}{2}\langle \mathbf{x} - \mathbf{x}', \mathbf{x} - \mathbf{x}'\rangle\right]$$

$$= \exp\left[-\frac{1}{2}(\langle \mathbf{x}, \mathbf{x} - \mathbf{x}'\rangle - \langle \mathbf{x}', \mathbf{x} - \mathbf{x}'\rangle)\right]$$

$$= \exp\left[-\frac{1}{2}(\langle \mathbf{x}, \mathbf{x} - \mathbf{x}'\rangle - \langle \mathbf{x}', \mathbf{x} - \mathbf{x}'\rangle)\right]$$

$$= \exp\left[-\frac{1}{2}(\langle \mathbf{x}, \mathbf{x}\rangle - \langle \mathbf{x}, \mathbf{x}'\rangle - \langle \mathbf{x}', \mathbf{x}\rangle + \langle \mathbf{x}', \mathbf{x}'\rangle)\right]$$

$$= \exp\left[-\frac{1}{2}(\langle \mathbf{x}, \mathbf{x}\rangle - \langle \mathbf{x}, \mathbf{x}'\rangle - \langle \mathbf{x}', \mathbf{x}\rangle + \langle \mathbf{x}', \mathbf{x}'\rangle)\right]$$

$$= \exp\left[-\frac{1}{2}(\|\mathbf{x}\|^2 + \|\mathbf{x}'\|^2 - 2\langle \mathbf{x}, \mathbf{x}'\rangle)\right]$$

$$= \exp\left[-\frac{1}{2}\|\mathbf{x}\|^2 - \frac{1}{2}\|\mathbf{x}'\|^2\right]\exp\left[-\frac{1}{2} - 2\langle \mathbf{x}, \mathbf{x}'\rangle\right]$$

$$= Ce^{\langle \mathbf{x}, \mathbf{x}'\rangle}$$

$C := \exp\left[-\frac{1}{2}\|\mathbf{x}\|^2 - \frac{1}{2}\|\mathbf{x}'\|^2\right]$ is a constant

$$= C\sum_{n=0}^{\infty}\frac{\langle \mathbf{x}, \mathbf{x}'\rangle^n}{n!}$$

Taylor expansion of e^x

$$= C\sum_{n=0}^{\infty}\frac{K_{\text{poly(n)}}(\mathbf{x}, \mathbf{x}')}{n!}$$

We see that the RBF kernel is formed by taking an infinite sum over polynomial kernels.

http://pages.cs.wisc.edu/~matthewb/pages/notes/pdf/svms/RBFKernel.pdf