

# CSCI567 Machine Learning (Spring 2018)

Michael Shindler

Lecture 10 on February 12, 2018

# Outline

- 1 Administration
- 2 Kernel methods

# Outline

- 1 Administration
- 2 Kernel methods

# Administrative stuff

- Problem Set / Programming Assignments 2 released
- If you submit PA1 late, **do not force push**
- When asking a question on Piazza:
  - Check if question already asked
  - Consult sources first as appropriate (e.g., syllabus)

# Outline

## 1 Administration

## 2 Kernel methods

- Motivation
- Kernel matrix and kernel functions
- Kernelized machine learning methods

# Motivation

**How to choose nonlinear basis function for classification and regression?**

$$\mathbf{w}^T \phi(\mathbf{x})$$

where  $\phi(\cdot)$  maps the original feature vector  $\mathbf{x}$  to a  $M$ -dimensional *new* feature vector.

# Motivation

## How to choose nonlinear basis function for classification and regression?

$$w^T \phi(x)$$

where  $\phi(\cdot)$  maps the original feature vector  $x$  to a  $M$ -dimensional *new* feature vector.

- We have seen the neural network approach: it learns features from data together with the classifier or regression function
- Any other approaches?

# Kernel methods

In this lecture, we will show that we can sidestep the issue of choosing which  $\phi(\cdot)$  to use — instead, we will choose *equivalently* a *kernel function*.

We will motivate our approach by re-examining something we have already known.



# Regularized least square

Our objective is to minimize the following regularized residual sum of squares

$$J(\mathbf{w}) = \frac{1}{2} \sum_n (y_n - \mathbf{w}^T \phi(\mathbf{x}_n))^2 + \frac{\lambda}{2} \|\mathbf{w}\|_2^2$$

# Regularized least square

Our objective is to minimize the following regularized residual sum of squares

$$J(\mathbf{w}) = \frac{1}{2} \sum_n (y_n - \mathbf{w}^T \phi(\mathbf{x}_n))^2 + \frac{\lambda}{2} \|\mathbf{w}\|_2^2$$

Its solution  $\mathbf{w}^*$  is given by

$$\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} = \sum_n (y_n - \mathbf{w}^T \phi(\mathbf{x}_n))(-\phi(\mathbf{x}_n)) + \lambda \mathbf{w} = 0$$

We can certainly solve  $\mathbf{w}^*$  now (as before).

# Optimal solution to regularized least square (see Lecture 4 too)

$$\mathbf{w}^* = (\Phi^T \Phi + \lambda I)^{-1} \Phi^T \mathbf{y}$$

where the solution is formulated in terms of design matrix  $\Phi$  and the target vector  $\mathbf{y}$

$$\Phi = \begin{pmatrix} \phi(\mathbf{x}_1)^T \\ \phi(\mathbf{x}_2)^T \\ \vdots \\ \phi(\mathbf{x}_N)^T \end{pmatrix} \in \mathbb{R}^{N \times M}$$

## Alternative solution

While the previous steps are valid and good, we will take a different route by not rushing into solving it – but instead, examining the structure of the solution

$$\lambda \mathbf{w} = \sum_n (y_n - \mathbf{w}^T \phi(\mathbf{x}_n)) \phi(\mathbf{x}_n)$$

# Optimal Solution

**The optimal parameter vector is a linear combination of features**

$$\mathbf{w}^* = \sum_n \frac{1}{\lambda} (y_n - \mathbf{w}^{*\top} \phi(\mathbf{x}_n)) \phi(\mathbf{x}_n) = \sum_n \alpha_n \phi(\mathbf{x}_n) = \Phi^\top \alpha$$

where we have designated  $\frac{1}{\lambda} (y_n - \mathbf{w}^{*\top} \phi(\mathbf{x}_n))$  as  $\alpha_n$ .

The design matrix  $\Phi$  is now transposed, thus is made of column vectors and is given by

$$\Phi^\top = (\phi(\mathbf{x}_1) \ \phi(\mathbf{x}_2) \ \cdots \ \phi(\mathbf{x}_N)) \in \mathbb{R}^{M \times N}$$

where  $M$  is the dimensionality of  $\phi(\mathbf{x})$ .

*Of course, we do not know what  $\alpha$  (the vector of all  $\alpha_n$ ) corresponds to  $\mathbf{w}^*$ !*

# Important observation

The optimal parameter vector is a linear combination of features

$$\mathbf{w}^* = \sum_n \alpha_n \mathbf{x}_n, \text{ or } \mathbf{w}^* = \sum_n \alpha_n \phi(\mathbf{x}_n)$$

- This is true for regularized linear regression
- This is true for perceptron
- This is true for many other types of algorithms.

But how to find  $\alpha_n$ ?

# Dual formulation

We substitute  $\mathbf{w}^* = \Phi^T \alpha$  into  $J(\mathbf{w})$ , and obtain the following function of  $\alpha$

$$J(\alpha) = \frac{1}{2} \alpha^T \Phi \Phi^T \Phi \Phi^T \alpha - (\Phi \Phi^T \mathbf{y})^T \alpha + \frac{\lambda}{2} \alpha^T \Phi \Phi^T \alpha$$

Before we show how  $J(\alpha)$  is derived, we make an important observation. We see repeated structures  $\Phi \Phi^T$ , to which we refer as *Gram matrix* or *kernel matrix*

$$\mathbf{K} = \Phi \Phi^T$$

$$= \begin{pmatrix} \phi(\mathbf{x}_1)^T \phi(\mathbf{x}_1) & \phi(\mathbf{x}_1)^T \phi(\mathbf{x}_2) & \cdots & \phi(\mathbf{x}_1)^T \phi(\mathbf{x}_N) \\ \phi(\mathbf{x}_2)^T \phi(\mathbf{x}_1) & \phi(\mathbf{x}_2)^T \phi(\mathbf{x}_2) & \cdots & \phi(\mathbf{x}_2)^T \phi(\mathbf{x}_N) \\ \cdots & \cdots & \cdots & \cdots \\ \phi(\mathbf{x}_N)^T \phi(\mathbf{x}_1) & \phi(\mathbf{x}_N)^T \phi(\mathbf{x}_2) & \cdots & \phi(\mathbf{x}_N)^T \phi(\mathbf{x}_N) \end{pmatrix} \in \mathbb{R}^{N \times N}$$

# Examples of kernel matrix

Let us assume we have 3 data points

$$x_1 = -1, x_2 = 0, x_3 = 1$$

And we have the following nonlinear mapping

$$\phi(x) = \begin{pmatrix} 1 \\ x \\ e^x \end{pmatrix}$$

How to compute the kernel matrix?



# Calculation of the mapping

$$x_1 \rightarrow \phi(x_1) = \begin{pmatrix} 1 \\ -1 \\ e^{-1} \end{pmatrix}, x_2 \rightarrow \phi(x_2) = \begin{pmatrix} 1 \\ 0 \\ e^0 \end{pmatrix}, x_3 \rightarrow \phi(x_3) = \begin{pmatrix} 1 \\ 1 \\ e^1 \end{pmatrix}$$

## Kernel matrix

$$\mathbf{K} = \begin{pmatrix} K_{11} & K_{12} & K_{13} \\ K_{21} & K_{22} & K_{23} \\ K_{31} & K_{32} & K_{33} \end{pmatrix}$$

where, for example,

$$K_{13} = \phi(x_1)^T \phi(x_3) = 1 \times 1 + (-1) \times 1 + e^{-1} \times e^1 = 1$$

# Properties of the matrix $K$

- Symmetric

$$K_{mn} = \phi(\mathbf{x}_m)^T \phi(\mathbf{x}_n) = \phi(\mathbf{x}_n)^T \phi(\mathbf{x}_m) = K_{nm}$$

- Positive semidefinite: for any vector  $\mathbf{a}$

$$\mathbf{a}^T K \mathbf{a} = (\Phi^T \mathbf{a})^T (\Phi^T \mathbf{a}) \geq 0$$

# The derivation of $J(\alpha)$

Derivation in the following is supplementary

$$\begin{aligned} J(w) &= \frac{1}{2} \sum_n (y_n - w^T \phi(x_n))^2 + \frac{\lambda}{2} \|w\|_2^2 \\ &= \frac{1}{2} \|y - \Phi w\|_2^2 + \frac{\lambda}{2} \|w\|_2^2 \\ &= \frac{1}{2} \|y - \Phi \Phi^T \alpha\|_2^2 + \frac{\lambda}{2} \|\Phi^T \alpha\|_2^2 \\ &= \frac{1}{2} \|y - K \alpha\|_2^2 + \frac{\lambda}{2} \alpha^T \Phi \Phi^T \alpha \\ &= \frac{1}{2} \alpha^T K^T K \alpha - y^T K \alpha + \frac{\lambda}{2} \alpha^T K \alpha \\ &= \frac{1}{2} \alpha^T K^2 \alpha - (Ky)^T \alpha + \frac{\lambda}{2} \alpha^T K \alpha = J(\alpha) \end{aligned}$$

where we have used the property that  $K$  is symmetric.

# Optimal $\alpha$

$$\frac{\partial J(\alpha)}{\partial \alpha} = K^2 \alpha - K y + \lambda K \alpha = 0$$

which leads to (assuming that  $K$  is invertible)

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

# Optimal $\alpha$

$$\frac{\partial J(\alpha)}{\partial \alpha} = K^2 \alpha - K y + \lambda K \alpha = 0$$

which leads to (assuming that  $K$  is invertible)

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

From this, we can compute the parameter vector

$$w^* = \Phi^T \alpha^* = \Phi^T (K + \lambda I)^{-1} y$$

# Compare to the old solution

## Previous approach

$$\mathbf{w}^* = (\Phi^T \Phi + \lambda I)^{-1} \Phi^T \mathbf{y}$$

## Now

$$\mathbf{w}^* = \Phi^T (\Phi \Phi^T + \lambda I)^{-1} \mathbf{y}$$

## Key difference

Kernel matrix  $\mathbf{K} = \Phi \Phi^T$  is not the same as the second-moment (covariance) matrix  $\mathbf{C} = \Phi^T \Phi$

- 1  $\mathbf{C}$  has a size of  $M \times M$  while  $\mathbf{K}$  is  $N \times N$ .
- 2 When  $N \leq D$ , using  $\mathbf{K}$  is more computationally advantageous

# Has this been helping?

Computing  $\alpha^*$  need only  $K$

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

Computing  $w^*$  need to know  $\Phi$

$$w^* = \Phi^T (K + \lambda I)^{-1} y$$

What is the difference?

To compute  $K$ , the exact form of  $\phi(\cdot)$  is not essential — as long as we know how to get inner products  $\phi(x_m)^T \phi(x_n)$ .

Now, I am asking you to believe me that indeed we can compute  $K$  without knowing what  $\phi(\cdot)$  is at all!

This is a “trick” known as *kernel trick*.

In fact, we really do not need to know  $w$

**Because computing prediction needs only inner products too!**

Suppose we need to make a prediction on a new data point  $x$ , we thus compute

$$(w^*)^T \phi(x) = y^T (K + \lambda I)^{-1} \Phi^T \phi(x)$$



In fact, we really do not need to know  $w$

**Because computing prediction needs only inner products too!**

Suppose we need to make a prediction on a new data point  $x$ , we thus compute

$$\begin{aligned}(w^*)^T \phi(x) &= y^T (K + \lambda I)^{-1} \Phi^T \phi(x) \\ &= y^T (K + \lambda I)^{-1} \begin{pmatrix} \phi(x_1)^T \phi(x) \\ \phi(x_2)^T \phi(x) \\ \vdots \\ \phi(x_N)^T \phi(x) \end{pmatrix} = y^T (K + \lambda I)^{-1} k_x\end{aligned}$$

where we have used the property that  $(K + \lambda I)^{-1}$  is symmetric (as  $K$  is) and use  $k_x$  as a shorthand notation for the column vector.

Note that, to make a prediction, once again, we *only need to know how to get  $\phi(x_n)^T \phi(x)$* .

# Summary

We can design algorithms such that we need only the inner products of transformed features between any pair of data points

$$\phi(\mathbf{x}_n)^T \phi(\mathbf{x}_m)$$

Our next step is to show that in fact, we can get the those inner products by computing

$$K(\mathbf{x}_m, \mathbf{x}_n)$$

which is a type of special functions over the original feature space of  $\mathbf{x}$ , without even knowing what  $\phi(\mathbf{x})$  is!

# Inner products between features

Due to their central roles, let us examine more closely the inner products  $\phi(\mathbf{x}_m)^T \phi(\mathbf{x}_n)$  for a pair of data points  $\mathbf{x}_m$  and  $\mathbf{x}_n$ .

**Polynomial-based nonlinear basis functions** consider the following  $\phi(\mathbf{x})$ :

$$\phi : \mathbf{x} \rightarrow \phi(\mathbf{x}) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix}$$

## Inner products between features

Due to their central roles, let us examine more closely the inner products  $\phi(\mathbf{x}_m)^T \phi(\mathbf{x}_n)$  for a pair of data points  $\mathbf{x}_m$  and  $\mathbf{x}_n$ .

**Polynomial-based nonlinear basis functions** consider the following  $\phi(\mathbf{x})$ :

$$\phi : \mathbf{x} \rightarrow \phi(\mathbf{x}) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix}$$

This gives rise to an inner product in a special form,

$$\begin{aligned} \phi(\mathbf{x}_m)^T \phi(\mathbf{x}_n) &= x_{m1}^2 x_{n1}^2 + 2x_{m1}x_{m2}x_{n1}x_{n2} + x_{m2}^2 x_{n2}^2 \\ &= (x_{m1}x_{n1} + x_{m2}x_{n2})^2 = (\mathbf{x}_m^T \mathbf{x}_n)^2 \end{aligned}$$

Namely, the inner product can be computed by a function  $(\mathbf{x}_m^T \mathbf{x}_n)^2$  defined in terms of the original features, *without knowing  $\phi(\cdot)$* .

## A more challenging example

Consider the following mapping, which is parameterized by a parameter

$$\psi_{\theta}(\mathbf{x}) = \begin{pmatrix} \cos(\theta x_1) \\ \sin(\theta x_1) \\ \cos(\theta x_2) \\ \sin(\theta x_2) \end{pmatrix}$$

The inner product for transformed  $\mathbf{x}_m$  and  $\mathbf{x}_n$  is thus,

$$\psi_{\theta}(\mathbf{x}_m)^T \psi_{\theta}(\mathbf{x}_n) = \cos(\theta(x_{m1} - x_{n1})) + \cos(\theta(x_{m2} - x_{n2}))$$

Note that, once again, *the inner product can be computed alternatively with the function in the right-hand-side, which depends on the original features only*. We will make it further more interesting.

## Concatenating into a long feature vector

Now, consider  $(L + 1)$   $\theta$ s, drawn from  $[0 \ 2\pi]$  evenly, and make another mapping

$$\phi_L(\mathbf{x}) = \begin{pmatrix} \psi_0(\mathbf{x}) \\ \psi_{\frac{2\pi}{L}}(\mathbf{x}) \\ \psi_{2\frac{2\pi}{L}}(\mathbf{x}) \\ \vdots \\ \psi_{L\frac{2\pi}{L}}(\mathbf{x}) \end{pmatrix}$$

## Concatenating into a long feature vector

Now, consider  $(L + 1)$   $\theta$ s, drawn from  $[0 \ 2\pi]$  evenly, and make another mapping

$$\phi_L(\mathbf{x}) = \begin{pmatrix} \psi_0(\mathbf{x}) \\ \psi_{\frac{2\pi}{L}}(\mathbf{x}) \\ \psi_{2\frac{2\pi}{L}}(\mathbf{x}) \\ \vdots \\ \psi_{L\frac{2\pi}{L}}(\mathbf{x}) \end{pmatrix}$$

What is the inner product?

$$\begin{aligned} \phi_L(\mathbf{x}_m)^T \phi_L(\mathbf{x}_n) &= \sum_{l=0}^L \psi_{l\frac{2\pi}{L}}(\mathbf{x}_m)^T \psi_{l\frac{2\pi}{L}}(\mathbf{x}_n) \\ &= \sum_{l=0}^L \cos\left(l\frac{2\pi}{L}(x_{m1} - x_{n1})\right) + \cos\left(l\frac{2\pi}{L}(x_{m2} - x_{n2})\right) \end{aligned}$$

## What if $L = +\infty$ ?

Instead of summing up  $(L + 1)$  terms, we will be integrating

$$\begin{aligned}
 \phi_{\infty}(\mathbf{x}_m)^T \phi_{\infty}(\mathbf{x}_n) &= \lim_{L \rightarrow +\infty} \phi_L(\mathbf{x}_m)^T \phi_L(\mathbf{x}_n) \\
 &= \int_0^{2\pi} \cos(\theta(x_{m1} - x_{n1})) + \cos(\theta(x_{m2} - x_{n2})) d\theta \\
 &= 1 - \frac{\sin(2\pi(x_{m1} - x_{n1}))}{x_{m1} - x_{n1}} + 1 - \frac{\sin(2\pi(x_{m2} - x_{n2}))}{x_{m2} - x_{n2}}
 \end{aligned}$$

While as before, the right-hand-side depends on only the original features. It actually computes the inner product of two *infinite-dimensional* feature vectors! (Since  $L \rightarrow +\infty$ , the number of  $\psi_{l\frac{2\pi}{L}}(\mathbf{x})$  is infinite, hence the dimensionality of  $\phi_{\infty}(\mathbf{x})$ .)

*In other words, while we cannot write down every dimension of  $\phi_{\infty}(\mathbf{x})$ , we can compute its inner product easily using a function defined on the original finite feature space.*



# Kernel functions

**Definition:** a (positive semidefinite) kernel function  $k(\cdot, \cdot)$  is a bivariate function that satisfies the following properties. For any  $\mathbf{x}_m$  and  $\mathbf{x}_n$ ,

$$k(\mathbf{x}_m, \mathbf{x}_n) = k(\mathbf{x}_n, \mathbf{x}_m) \text{ and } k(\mathbf{x}_m, \mathbf{x}_n) = \phi(\mathbf{x}_m)^T \phi(\mathbf{x}_n)$$

for *some* function  $\phi(\cdot)$ .

# Kernel functions

**Definition:** a (positive semidefinite) kernel function  $k(\cdot, \cdot)$  is a bivariate function that satisfies the following properties. For any  $\mathbf{x}_m$  and  $\mathbf{x}_n$ ,

$$k(\mathbf{x}_m, \mathbf{x}_n) = k(\mathbf{x}_n, \mathbf{x}_m) \text{ and } k(\mathbf{x}_m, \mathbf{x}_n) = \phi(\mathbf{x}_m)^T \phi(\mathbf{x}_n)$$

for *some* function  $\phi(\cdot)$ .

## Examples we have seen

$$k(\mathbf{x}_m, \mathbf{x}_n) = (\mathbf{x}_m^T \mathbf{x}_n)^2$$

$$k(\mathbf{x}_m, \mathbf{x}_n) = 2 - \frac{\sin(2\pi(x_{m1} - x_{n1}))}{x_{m1} - x_{n1}} - \frac{\sin(2\pi(x_{m2} - x_{n2}))}{x_{m2} - x_{n2}}$$

# Kernel functions

**Definition:** a (positive semidefinite) kernel function  $k(\cdot, \cdot)$  is a bivariate function that satisfies the following properties. For any  $\mathbf{x}_m$  and  $\mathbf{x}_n$ ,

$$k(\mathbf{x}_m, \mathbf{x}_n) = k(\mathbf{x}_n, \mathbf{x}_m) \text{ and } k(\mathbf{x}_m, \mathbf{x}_n) = \phi(\mathbf{x}_m)^T \phi(\mathbf{x}_n)$$

for *some* function  $\phi(\cdot)$ .

## Examples we have seen

$$k(\mathbf{x}_m, \mathbf{x}_n) = (\mathbf{x}_m^T \mathbf{x}_n)^2$$

$$k(\mathbf{x}_m, \mathbf{x}_n) = 2 - \frac{\sin(2\pi(x_{m1} - x_{n1}))}{x_{m1} - x_{n1}} - \frac{\sin(2\pi(x_{m2} - x_{n2}))}{x_{m2} - x_{n2}}$$

## Examples that are not kernels

$$k(\mathbf{x}_m, \mathbf{x}_n) = \|\mathbf{x}_m - \mathbf{x}_n\|_2^2$$

are not our desired kernel function as it cannot be written as inner products between two vectors.

# Conditions for being a positive semidefinite kernel function

**Mercer theorem** (loosely), a bivariate function  $k(\cdot, \cdot)$  is a positive semidefinite kernel function, if and only if, for *any*  $N$  and *any*  $\mathbf{x}_1, \mathbf{x}_2, \dots$ , and  $\mathbf{x}_N$ , the matrix

$$\mathbf{K} = \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & k(\mathbf{x}_1, \mathbf{x}_2) & \cdots & k(\mathbf{x}_1, \mathbf{x}_N) \\ k(\mathbf{x}_2, \mathbf{x}_1) & k(\mathbf{x}_2, \mathbf{x}_2) & \cdots & k(\mathbf{x}_2, \mathbf{x}_N) \\ \vdots & \vdots & \vdots & \vdots \\ k(\mathbf{x}_N, \mathbf{x}_1) & k(\mathbf{x}_N, \mathbf{x}_2) & \cdots & k(\mathbf{x}_N, \mathbf{x}_N) \end{pmatrix}$$

is positive semidefinite. We also refer  $k(\cdot, \cdot)$  as a positive semidefinite kernel.

# Flashback: why using kernel functions?

without specifying  $\phi(\cdot)$ , the kernel matrix

$$K = \begin{pmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & k(\mathbf{x}_1, \mathbf{x}_2) & \cdots & k(\mathbf{x}_1, \mathbf{x}_N) \\ k(\mathbf{x}_2, \mathbf{x}_1) & k(\mathbf{x}_2, \mathbf{x}_2) & \cdots & k(\mathbf{x}_2, \mathbf{x}_N) \\ \vdots & \vdots & \vdots & \vdots \\ k(\mathbf{x}_N, \mathbf{x}_1) & k(\mathbf{x}_N, \mathbf{x}_2) & \cdots & k(\mathbf{x}_N, \mathbf{x}_N) \end{pmatrix}$$

is exactly the same as

$$\begin{aligned} K &= \Phi \Phi^T \\ &= \begin{pmatrix} \phi(\mathbf{x}_1)^T \phi(\mathbf{x}_1) & \phi(\mathbf{x}_1)^T \phi(\mathbf{x}_2) & \cdots & \phi(\mathbf{x}_1)^T \phi(\mathbf{x}_N) \\ \phi(\mathbf{x}_2)^T \phi(\mathbf{x}_1) & \phi(\mathbf{x}_2)^T \phi(\mathbf{x}_2) & \cdots & \phi(\mathbf{x}_2)^T \phi(\mathbf{x}_N) \\ \cdots & \cdots & \cdots & \cdots \\ \phi(\mathbf{x}_N)^T \phi(\mathbf{x}_1) & \phi(\mathbf{x}_N)^T \phi(\mathbf{x}_2) & \cdots & \phi(\mathbf{x}_N)^T \phi(\mathbf{x}_N) \end{pmatrix} \end{aligned}$$

# Examples of kernel functions

## Polynomial kernel function with degree of $d$

$$k(\mathbf{x}_m, \mathbf{x}_n) = (\mathbf{x}_m^T \mathbf{x}_n + c)^d$$

for  $c \geq 0$  and  $d$  is a positive integer.

# Examples of kernel functions

## Polynomial kernel function with degree of $d$

$$k(\mathbf{x}_m, \mathbf{x}_n) = (\mathbf{x}_m^T \mathbf{x}_n + c)^d$$

for  $c \geq 0$  and  $d$  is a positive integer.

## Gaussian kernel, RBF kernel, or Gaussian RBF kernel

$$k(\mathbf{x}_m, \mathbf{x}_n) = e^{-\|\mathbf{x}_m - \mathbf{x}_n\|_2^2 / 2\sigma^2}$$

# Examples of kernel functions

## Polynomial kernel function with degree of $d$

$$k(\mathbf{x}_m, \mathbf{x}_n) = (\mathbf{x}_m^T \mathbf{x}_n + c)^d$$

for  $c \geq 0$  and  $d$  is a positive integer.

## Gaussian kernel, RBF kernel, or Gaussian RBF kernel

$$k(\mathbf{x}_m, \mathbf{x}_n) = e^{-\|\mathbf{x}_m - \mathbf{x}_n\|_2^2 / 2\sigma^2}$$

Most of those kernels have parameters to be tuned:  $d$ ,  $c$ ,  $\sigma^2$ , etc. They are hyper parameters and are often tuned on holdout data or with cross-validation.



## Why $\|\mathbf{x}_m - \mathbf{x}_n\|_2^2$ is not a positive semidefinite kernel?

**Use the definition** of positive semidefinite kernel function. We choose  $N = 2$ , and compute the matrix

$$\mathbf{K} = \begin{pmatrix} 0 & \|\mathbf{x}_1 - \mathbf{x}_2\|_2^2 \\ \|\mathbf{x}_1 - \mathbf{x}_2\|_2^2 & 0 \end{pmatrix}$$

This matrix cannot be positive semidefinite as it has both *negative* and positive eigenvalues (the sum of the diagonal elements is called the trace of a matrix, which equals to the sum of the matrix's eigenvalues. In our case, the trace is zero.)

# There are infinite numbers of kernels to use!

## Rules of composing kernels (this is just a partial list)

- if  $k(\mathbf{x}_m, \mathbf{x}_n)$  is a kernel, then  $ck(\mathbf{x}_m, \mathbf{x}_n)$  is also if  $c > 0$ .
- if both  $k_1(\mathbf{x}_m, \mathbf{x}_n)$  and  $k_2(\mathbf{x}_m, \mathbf{x}_n)$  are kernels, then  $\alpha k_1(\mathbf{x}_m, \mathbf{x}_n) + \beta k_2(\mathbf{x}_m, \mathbf{x}_n)$  are also if  $\alpha, \beta \geq 0$
- if both  $k_1(\mathbf{x}_m, \mathbf{x}_n)$  and  $k_2(\mathbf{x}_m, \mathbf{x}_n)$  are kernels, then  $k_1(\mathbf{x}_m, \mathbf{x}_n)k_2(\mathbf{x}_m, \mathbf{x}_n)$  are also.
- if  $k(\mathbf{x}_m, \mathbf{x}_n)$  is a kernel, then  $e^{k(\mathbf{x}_m, \mathbf{x}_n)}$  is also.
- ...

In practice, using which kernel, or which kernels to compose a new kernel, remains somewhat as “art,” though most people will start with polynomial and Gaussian RBF kernels.

# Kernelization trick

Many learning methods depend on computing *inner products* between features — we have seen the example of regularized least squares. For those methods, we can use a kernel function in the place of the inner products, i.e., “*kernerlizing*” the methods, thus, introducing nonlinear features/basis.

We will present one more to illustrate this “trick” by kernerlizing nearest neighbor classifier.

When we talk about support vector machines next lecture, we will see the trick one more time.

## Kernelized nearest neighbor classifier

In nearest neighbor classifier, the most important quantity to compute is the (squared) distance between two data points  $\mathbf{x}_m$  and  $\mathbf{x}_n$

$$d(\mathbf{x}_m, \mathbf{x}_n) = \|\mathbf{x}_m - \mathbf{x}_n\|_2^2 = \mathbf{x}_m^T \mathbf{x}_m + \mathbf{x}_n^T \mathbf{x}_n - 2\mathbf{x}_m^T \mathbf{x}_n$$

## Kernelized nearest neighbor classifier

In nearest neighbor classifier, the most important quantity to compute is the (squared) distance between two data points  $\mathbf{x}_m$  and  $\mathbf{x}_n$

$$d(\mathbf{x}_m, \mathbf{x}_n) = \|\mathbf{x}_m - \mathbf{x}_n\|_2^2 = \mathbf{x}_m^T \mathbf{x}_m + \mathbf{x}_n^T \mathbf{x}_n - 2\mathbf{x}_m^T \mathbf{x}_n$$

We replace all the inner products in the distance with a kernel function  $k(\cdot, \cdot)$ , arriving at the kernelized distance

$$d^{\text{KERNEL}}(\mathbf{x}_m, \mathbf{x}_n) = k(\mathbf{x}_m, \mathbf{x}_m) + k(\mathbf{x}_n, \mathbf{x}_n) - 2k(\mathbf{x}_m, \mathbf{x}_n)$$

## Kernelized nearest neighbor classifier

In nearest neighbor classifier, the most important quantity to compute is the (squared) distance between two data points  $\mathbf{x}_m$  and  $\mathbf{x}_n$

$$d(\mathbf{x}_m, \mathbf{x}_n) = \|\mathbf{x}_m - \mathbf{x}_n\|_2^2 = \mathbf{x}_m^T \mathbf{x}_m + \mathbf{x}_n^T \mathbf{x}_n - 2\mathbf{x}_m^T \mathbf{x}_n$$

We replace all the inner products in the distance with a kernel function  $k(\cdot, \cdot)$ , arriving at the kernelized distance

$$d^{\text{KERNEL}}(\mathbf{x}_m, \mathbf{x}_n) = k(\mathbf{x}_m, \mathbf{x}_m) + k(\mathbf{x}_n, \mathbf{x}_n) - 2k(\mathbf{x}_m, \mathbf{x}_n)$$

The distance is equivalent to compute the distance between  $\phi(\mathbf{x}_m)$  and  $\phi(\mathbf{x}_n)$

$$d^{\text{KERNEL}}(\mathbf{x}_m, \mathbf{x}_n) = d(\phi(\mathbf{x}_m), \phi(\mathbf{x}_n))$$

where the  $\phi(\cdot)$  is the nonlinear mapping function implied by the kernel function.

## Kernelized nearest neighbor classifier

In nearest neighbor classifier, the most important quantity to compute is the (squared) distance between two data points  $\mathbf{x}_m$  and  $\mathbf{x}_n$

$$d(\mathbf{x}_m, \mathbf{x}_n) = \|\mathbf{x}_m - \mathbf{x}_n\|_2^2 = \mathbf{x}_m^T \mathbf{x}_m + \mathbf{x}_n^T \mathbf{x}_n - 2\mathbf{x}_m^T \mathbf{x}_n$$

We replace all the inner products in the distance with a kernel function  $k(\cdot, \cdot)$ , arriving at the kernelized distance

$$d^{\text{KERNEL}}(\mathbf{x}_m, \mathbf{x}_n) = k(\mathbf{x}_m, \mathbf{x}_m) + k(\mathbf{x}_n, \mathbf{x}_n) - 2k(\mathbf{x}_m, \mathbf{x}_n)$$

The distance is equivalent to compute the distance between  $\phi(\mathbf{x}_m)$  and  $\phi(\mathbf{x}_n)$

$$d^{\text{KERNEL}}(\mathbf{x}_m, \mathbf{x}_n) = d(\phi(\mathbf{x}_m), \phi(\mathbf{x}_n))$$

where the  $\phi(\cdot)$  is the nonlinear mapping function implied by the kernel function. The nearest neighbor of a point  $\mathbf{x}$  is thus found with

$$\arg \min_n d^{\text{KERNEL}}(\mathbf{x}, \mathbf{x}_n)$$

# Take-home exercise

## You have seen examples of kernelizing

- linear regression
- nearest neighbor

## But can you kernelize the following?

- Logistic (or multinomial logistic) regression

*You are welcome to Google search the answers after you spend sometime (say about 1 hour or so) on this exercise.*