EECS 545: Machine Learning

Lecture 8. Kernel methods

Honglak Lee and Michał Dereziński

02/02/2022

Logistics/Announcements

- HW 1 grade will be released on 02/05
- HW 2 due on 02/08
- Project Proposal due
  - No point deduction till 02/04
  - 1 point per day from 02/05 onwards
- CPU/GPU credits on Great Lakes - Check Piazza
- Convex Optimization review on Friday

Outline

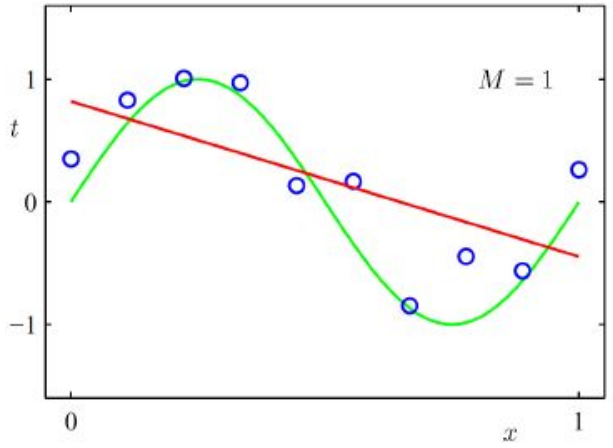
- Kernel methods: Motivation
- Kernel functions
- Kernel trick
- Kernel trick for linear regression
- Constructing kernels
- Kernel regression

Linear regression

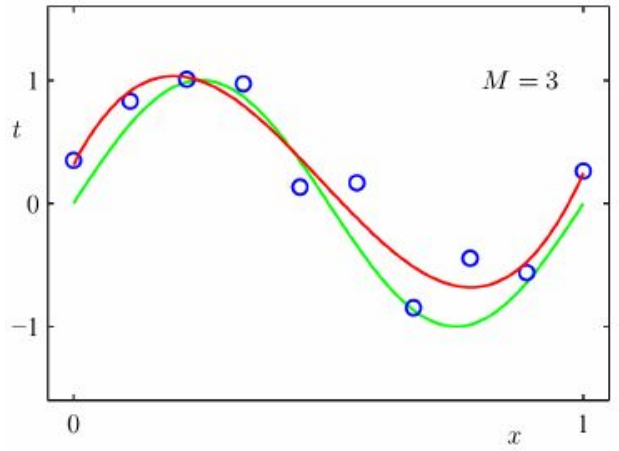
- Example: 1D regression
  - one input x, one output h(x)
- Linear model $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$ can only produce straight lines through origin
- Not very flexible/powerful
- How do we deal with this?

Feature mappings

- Replace $x \longrightarrow (1, x)$



- Replace $x \longrightarrow (1, x, x^2, x^3)$



Linear regression with (nonlinear) features

- Linear regression model

$$h(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \phi(\mathbf{x}) = \sum_{j=0}^{M} w_j \phi_j(\mathbf{x})$$

- Least-squares with L2 regression

$$J(\mathbf{w}) = \frac{1}{2} \sum_{n=0}^{N} (\mathbf{w}^T \phi(\mathbf{x}_n) - y_n)^2 + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w}$$

- Closed form solution:

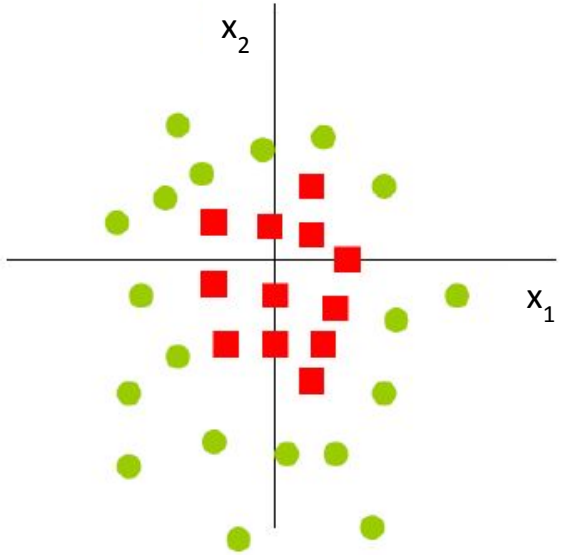
$$\mathbf{w} = (\Phi^T \Phi + \lambda \mathbf{I})^{-1} \mathbf{y}$$

This is nice, but…

- What features to use?
- Computational complexity
  - Φ: N*M matrix
    - N: number of examples
    - M: number of features
  - Need to invert $\Phi^T \Phi$ (M*M) matrix
  - Computational complexity scales with M^3

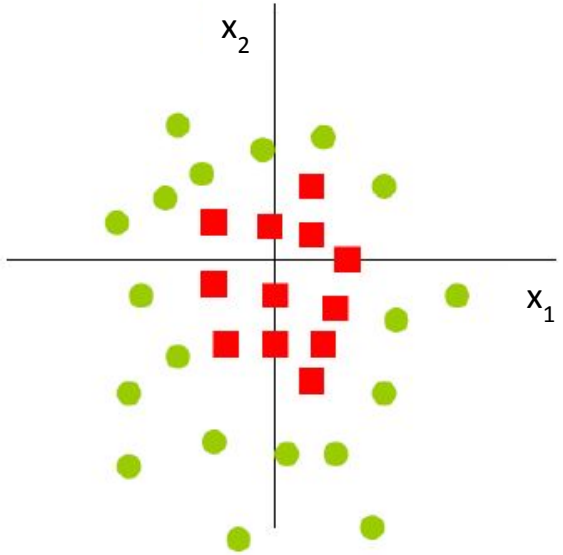
Linear Classifiers

- No linear separating plane
- Linear classifiers not very flexible/powerful
- Can we do better?



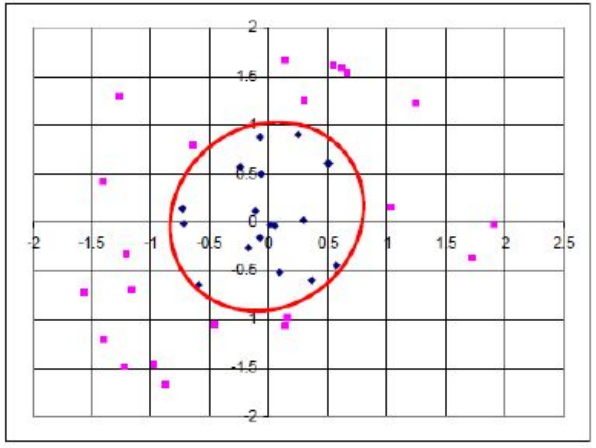
Linear classifiers with nonlinear features

- Add distance to origin $(x_1^2 + x_2^2)^{1/2}$ as a third feature
- Data now lives on a parabolic surface in 3D.
- Linear separation in 3D feature space.
- In original feature space, decision boundary is an ellipse

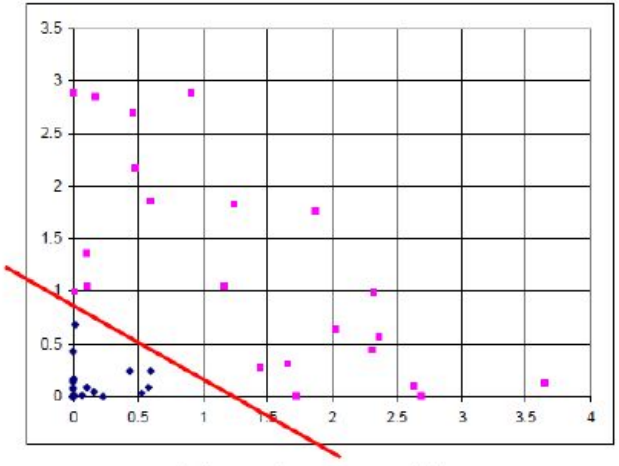


Linear Classifiers with nonlinear features

- Another way: Replace $(x_1, x_2) \rightarrow (x_1^2, x_2^2)$



Not linearly separable



Linearly separable

  - Different expansions make the problem solvable with linear methods.

Linear Classifiers with nonlinear features

- Data has been mapped to a new, higher dimensional space
- Alternative way to think about this: data still lives in original space, but the definition of *distance* or *inner product* has been changed

Classifiers with nonlinear features

- We have been mapping each data point x through a fixed non-linear mapping to get a feature vector $\phi(x)$
  - The feature vector extracts important properties from x.
  - E.g., polynomial combinations of the original features, up to some order
  - It may make many regression/classification problems easier.
- Unfortunately, the feature vector may be high-dimensional, even infinite-dimensional.
  - Problems: computational complexity

Kernels to the rescue (kernel trick)

- Embed data in a high dimensional space, and use simple models (linear relations) in this space.
- Use algorithms that do not need the coordinates of the embedded points, but only pairwise inner products
- Compute these inner products efficiently using a kernel

Kernel Functions

- A kernel function $k(\mathbf{x},\mathbf{x}')$ is intended to represent the similarity between **x** and **x'**.
- A popular way to express similarity is as the inner product of feature vectors:

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$$

- We *define* a kernel function $k(\mathbf{x},\mathbf{x}')$ as one that can be expressed as an inner product, but we may not need to compute it that way.
- This definition immediately leads to symmetricity of kernels:

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

Example: 2D input data

- Inner product between two vectors (x_1, x_2) and (z_1, z_2)

$$k(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T \mathbf{z} = x_1 z_1 + x_2 z_2$$

- Let's replace this by its square

$$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T \mathbf{z})^2 = x_1^2 z_1^2 + x_2^2 z_2^2 + 2 x_1 x_2 z_1 z_2$$

- This is the same as inner product between

$$(x_1^2, \sqrt{2} x_1 x_2, x_2^2) \text{ and } (z_1^2, \sqrt{2} z_1 z_2, z_2^2)$$

  - Or between

$$(x_1^2, x_1 x_2, x_1 x_2, x_2^2) \text{ and } (z_1^2, z_1 z_2, z_1 z_2, z_2^2)$$

  - Note: solution is not unique.

Example: 2D input data

- Consider higher-order polynomial of degree p:

$$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T \mathbf{z})^p = \left(\sum_{j=1}^{M} x_i z_i \right)^p$$

$$= \sum_{(j_1, j_2 \cdots j_M) : \sum_k j_k = p} \binom{p}{j_1, j_2 \cdots j_M} (x_1 z_1)^{j_1} (x_2 z_2)^{j_2} \cdots (x_M z_M)^{j_M}$$

- Feature mapping:

$$\phi(\mathbf{x}) = \left[\cdots, \sqrt{\binom{p}{j_1, j_2 \cdots j_M}}\ x_1^{j_1}\, x_2^{j_2} \ldots x_M^{j_M}, \cdots \right]^T$$

  - All monomials of degree p

Example: 2D input data

- Inhomogeneous polynomial up to degree p:

$$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T\mathbf{z} + c)^p = \left(\sum_{j=1}^{M} x_j z_j + c\right)^p, c > 0$$

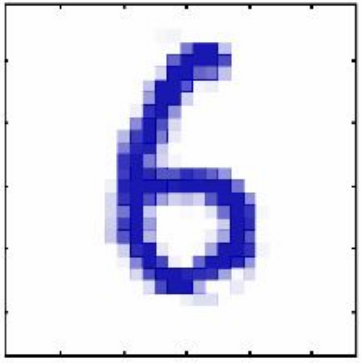
- Feature mapping:
  - All monomials of degree <= p

Example: handwritten digits images

- Take the pixel values and compute

$$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T\mathbf{z} + 1)^p$$

  - Here **x** is 28*28 = 784 dimensional



- You need to compute the inner product in the space of all monomials up to degree *p*.
- For dim(**x**)=784 and p=4 a 16-billion dimensional space!

Kernel trick

- Kernels allow you to achieve a high-dimensional feature space which is desirable for better separability for classes, i.e., classification performance.
- Crucially, we don't have to compute the high-dimensional feature explicitly, the inner product of the features are computed directly via the kernel function.
- Many algorithms can be expressed completely in terms of kernels k(x,x'), rather than other operations on x.
- In this case, you can replace one kernel with another, and get a new algorithm that works over a different domain.

$$k(\mathbf{x}, \mathbf{z}) = (\underbrace{\mathbf{x}^T \mathbf{z}}_{784\text{ dim}} + 1)^4 = \underbrace{\phi(\mathbf{x})^T \phi(\mathbf{z})}_{16\text{ billion dim}}$$

Kernel Trick

- The kernel trick represents the problem formulation and its solutions entirely in terms of kernels (this is called “dual representation”).
- The elements of the Gram matrix $\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^T$

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

- These represent the pairwise similarities among all the observed feature vectors.
  - We may be able to compute the kernels more efficiently than the feature vectors.

Kernel substitution

- To use the kernel trick, we must formulate (training and test) algorithms purely in terms of inner products between data points
- We *cannot* access the coordinates of points in the high-dimensional feature space
- This seems a huge limitation, but it turns out that quite a lot can be done

Example: distance

- Distance between samples can be expressed in inner products:

$$\begin{aligned}\|\phi(\mathbf{x}) - \phi(\mathbf{z})\|^2 &= \langle \phi(\mathbf{x}) - \phi(\mathbf{z}), \phi(\mathbf{x}) - \phi(\mathbf{z}) \rangle \\ &= \langle \phi(\mathbf{x}), \phi(\mathbf{x}) \rangle + \langle \phi(\mathbf{z}), \phi(\mathbf{z}) \rangle - 2\langle \phi(\mathbf{x}), \phi(\mathbf{z}) \\ &= k(\mathbf{x}, \mathbf{x}) + k(\mathbf{z}, \mathbf{z}) - 2k(\mathbf{x}, \mathbf{z})\end{aligned}$$

- So nothing stops you from doing *k*-nearest neighbor searches in high dimensional spaces

Example: mean

- Can you determine the mean of data in the mapped feature space through kernel operations only?
  - A: No, you cannot compute any point explicitly

Example: distance to the mean

- Mean of data points given by: $\phi_S = \frac{1}{N}\sum_{i=1}^{N}\phi(\mathbf{x}_i)$

- Distance to mean:

$$\begin{aligned}\|\phi(\mathbf{x}) - \phi_S\|^2 &= \langle\phi(\mathbf{x}), \phi(\mathbf{x})\rangle + \langle\phi_S, \phi_S\rangle - 2\langle\phi(\mathbf{x}), \phi_S\rangle \\ &= \kappa(\mathbf{x}, \mathbf{x}) + \frac{1}{N^2}\sum_{i,j=1}^{N}\kappa(\mathbf{x}_i, \mathbf{x}_j) - \frac{2}{N}\sum_{i=1}^{N}\kappa(\mathbf{x}, \mathbf{x}_i)\end{aligned}$$

Kernel Trick for linear regression

- Recall regression problems with error function

$$J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\{\mathbf{w}^T\phi\left(x_n\right) - y_n\right\}^2 + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

- *J*(w) is minimized at

$$\mathbf{w}_{ML} = \left(\lambda\mathbf{I} + \mathbf{\Phi}^T\mathbf{\Phi}\right)^{-1}\mathbf{\Phi}^T\mathbf{y}$$

- Recall the *N* x *M* design matrix that is central to this solution.
- We can approach the solution a different way

Recap: The Design Matrix

- The design matrix is an NxM matrix, applying
  - the M basis functions (M: number of columns)
  - to N data points (N: number of rows)

$$\Phi = \begin{pmatrix} \phi_0(\mathbf{x}_1) & \phi_1(\mathbf{x}_1) & \cdots & \phi_{M-1}(\mathbf{x}_1) \\ \phi_0(\mathbf{x}_2) & \phi_1(\mathbf{x}_2) & \cdots & \phi_{M-1}(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\mathbf{x}_N) & \phi_1(\mathbf{x}_N) & \cdots & \phi_{M-1}(\mathbf{x}_N) \end{pmatrix}$$

$$\Phi\mathbf{w} \approx \mathbf{y}$$

The Gram Matrix

- For regression, a key term is the *M*x*M* matrix

$$\mathbf{\Phi}^T\mathbf{\Phi}$$ "covariance"

- Here, we will use the NxN *Gram matrix*

$$\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^T$$ "pairwise similarity"

- Note that $K_{nm} = \phi(\mathbf{x}_n)^T\phi(\mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m)$
  - The pairwise similarities of all the data points in the training set.
- Note that kernel methods use only K, not Φ.

Kernel Trick for linear regression

- Another way to minimize *J*(w) is

$$\mathbf{w} = -\frac{1}{\lambda}\sum_{n=1}^{N}\left\{\mathbf{w}^T\phi\left(\mathbf{x}_n\right) - y_n\right\}\phi\left(\mathbf{x}_n\right) = \sum_{n=1}^{N} a_n\phi\left(\mathbf{x}_n\right) = \mathbf{\Phi}^T\mathbf{a}$$

  – where $a_n = -\frac{1}{\lambda}\left\{\mathbf{w}^T\phi\left(\mathbf{x}_n\right) - y_n\right\}$

- Let a be the **dual** parameter, instead of w.
- Transform *J*(w) to *J*(a) by substituting

$$\mathbf{w} = \mathbf{\Phi}^T\mathbf{a}$$

Kernel Trick for linear regression

- Objective function $J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\{\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n) - y_n\right\}^2 + \frac{\lambda}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$
- Substitute $\mathbf{w} = \boldsymbol{\Phi}^T\mathbf{a}$

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

$$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{K}\mathbf{a} - \mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{y} + \frac{1}{2}\mathbf{y}^{\mathrm{T}}\mathbf{y} + \frac{\lambda}{2}\mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{a}$$

- Solution/prediction: $\mathbf{a} = (\mathbf{K} + \lambda\mathbf{I}_N)^{-1}\mathbf{y}$

$$\begin{aligned} h(\mathbf{x}) &= \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}}\mathbf{w} = \boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}}\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{a} = (\boldsymbol{\Phi}\boldsymbol{\phi}(\mathbf{x}))^{\mathrm{T}}\mathbf{a} \\ &= \mathbf{k}(\mathbf{x})^{\mathrm{T}}\mathbf{a} = \sum_{j=1}^{N} a_n\mathbf{k}(\mathbf{x}_n, \mathbf{x}) \\ &= \mathbf{k}(\mathbf{x})^{\mathrm{T}}(\mathbf{K} + \lambda\mathbf{I}_N)^{-1}\mathbf{y} \end{aligned}$$

Kernel Trick for linear regression

- Transform *J*(w) to *J*(a) by using $\mathbf{w} = \mathbf{\Phi}^T \mathbf{a}$

  and the *Gram* matrix $\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^T$

- Find **a** to minimize *J*(a): $\mathbf{a} = (\mathbf{K} + \lambda \mathbf{I}_N)^{-1} \mathbf{y}$

- For predictions (for query point/test example **x**):

$$h(\mathbf{x}) = \boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}}\mathbf{w} = \boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}}\mathbf{\Phi}^{\mathrm{T}}\mathbf{a} = k(\mathbf{x})^{\mathrm{T}}\left(\mathbf{K} + \lambda \mathbf{I}_N\right)^{-1}\mathbf{y}$$

  – where

$$k(\mathbf{x}) = [k(\mathbf{x}_1, \mathbf{x}), \ldots, k(\mathbf{x}_N, \mathbf{x})]^T$$

- This method is called ***Kernel Ridge Regression***.

Primal versus Dual

- Primal: $\mathbf{w} = \left(\mathbf{\Phi}'\mathbf{\Phi} + \lambda \mathbf{I}_M\right)^{-1} \mathbf{\Phi}'\mathbf{y}$
- Dual: $\mathbf{a} = \left(\mathbf{K} + \lambda \mathbf{I}_N\right)^{-1} \mathbf{y}$
- Primal: invert *M* by *M* matrix *(M* = dim feature space), w vector of length *M*
  - cheaper because usually *N* > *M*, but you need to explicitly construct features.
- Dual: invert *N* by *N* matrix *(N* = number of data points)
  - can use the kernel trick (embed into very high dimensional feature space)
  - Use kernels *k*(x,x') to represent similarity.
  - Kernels can be defined over vectors, images, sequences, graphs, text, etc.

Constructing Valid Kernels

- One can do *kernel engineering* to create kernels for particular purposes, expressing different kinds of similarity.
- How do we verify that a kernel is valid?
- Three methods (for verification):
  - Direct construction with feature vectors
  - Mercer Theorem
  - Composition of kernels with pre-defined rules

Constructing Valid Kernels: Method 1

- Method 1: One way is to define the feature space mapping $\phi(\mathbf{x})$ and show that the kernel function represents the inner product of feature vectors:

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}') = \sum_{i=1}^{M} \phi_i(\mathbf{x}) \phi_i(\mathbf{x}')$$

Constructing Valid Kernels

- Suppose we define a kernel function directly, such as

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

- In 2D, we can explicitly identify the feature map

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

  - such that

$$k(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^T \phi(\mathbf{z})$$

- Explicit feature mappings can be very complex.
  - Kernels help us avoid that complexity.

Constructing Valid Kernels: Method 2

- A simpler way to test without having to construct ϕ(x)
- Use the necessary and sufficient condition (Mercer Theorem) that for a function k(x,x') to be a inner product (valid) kernel:
  - the Gram matrix **K**, whose elements are given by $k(\mathbf{x}_n, \mathbf{x}_m)$, should be positive semidefinite for all possible choices of the data set $\{x_n\}$
  - I.e., **K** is positive semidefinite:

$$\mathbf{a}^T\mathbf{K}\mathbf{a} = \sum_{i=1}^{N}\sum_{j=1}^{N} a_i K_{i,j} a_j \geq 0 \quad \forall \mathbf{a} \in \mathbb{R}^N$$

Constructing Valid Kernels: Method 3

- There are a number of axioms that help us construct new, more complex kernels, from simpler known kernels.
- For example,

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

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

$$k(\mathbf{x}, \mathbf{x}') = \exp(-||\mathbf{x} - \mathbf{x}'||^2 / 2\sigma^2)$$

  - Prove that these are valid kernels (homework)

Constructing Kernels

Given valid kernels $k_1(\mathbf{x}, \mathbf{x}')$ and $k_2(\mathbf{x}, \mathbf{x}')$, the following new kernels will also be valid:

$$
\begin{aligned}
k(\mathbf{x}, \mathbf{x}') &= c k_1(\mathbf{x}, \mathbf{x}') && (6.13)\\
k(\mathbf{x}, \mathbf{x}') &= f(\mathbf{x}) k_1(\mathbf{x}, \mathbf{x}') f(\mathbf{x}') && (6.14)\\
k(\mathbf{x}, \mathbf{x}') &= q\left(k_1(\mathbf{x}, \mathbf{x}')\right) && (6.15)\\
k(\mathbf{x}, \mathbf{x}') &= \exp\left(k_1(\mathbf{x}, \mathbf{x}')\right) && (6.16)\\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') && (6.17)\\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') k_2(\mathbf{x}, \mathbf{x}') && (6.18)\\
k(\mathbf{x}, \mathbf{x}') &= k_3\left(\boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}')\right) && (6.19)\\
k(\mathbf{x}, \mathbf{x}') &= \mathbf{x}^{\mathrm{T}} \mathbf{A} \mathbf{x}' && (6.20)\\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}'_a) + k_b(\mathbf{x}_b, \mathbf{x}'_b) && (6.21)\\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}'_a) k_b(\mathbf{x}_b, \mathbf{x}'_b) && (6.22)
\end{aligned}
$$

where $c > 0$ is a constant, $f(\cdot)$ is any function, $q(\cdot)$ is a polynomial with nonnegative coefficients, $\boldsymbol{\phi}(\mathbf{x})$ is a function from $\mathbf{x}$ to $\mathbb{R}^M$, $k_3(\cdot, \cdot)$ is a valid kernel in $\mathbb{R}^M$, $\mathbf{A}$ is a symmetric positive semidefinite matrix, $\mathbf{x}_a$ and $\mathbf{x}_b$ are variables (not necessarily disjoint) with $\mathbf{x} = (\mathbf{x}_a, \mathbf{x}_b)$, and k_a and k_b are valid kernel functions over their respective spaces.

Most popular kernels

- Simple Polynomial Kernel (terms of degree 2)

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

- Generalized Polynomial kernel - degree M

$$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T \mathbf{z} + c)^M, c > 0$$

- Gaussian Kernels

$$k(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{z}\|^2}{2\sigma^2}\right)$$

Gaussian kernel

- Not related to Gaussian pdf
- Translation invariant (depends only on distance between points)
- Corresponds to an infinitely dimensional space! (PRML ex6.11)

Kernel regression

Kernel regression

- Recall k-nearest neighbor regression:

$$h(\mathbf{x}) = \frac{1}{k} \sum_{(\mathbf{x}', y') \in kNN(\mathbf{x})} y'$$

- Kernel regression:

$$h(\mathbf{x}) = \frac{\sum_i \mathbf{k}(\mathbf{x}, \mathbf{x}_i) y_i}{\sum_j \mathbf{k}(\mathbf{x}, \mathbf{x}_j)} = \frac{1}{Z} \sum_i \mathbf{k}(\mathbf{x}, \mathbf{x}_i) y_i$$

$$\text{where} \quad Z = \sum_j \mathbf{k}(\mathbf{x}, \mathbf{x}_j)$$

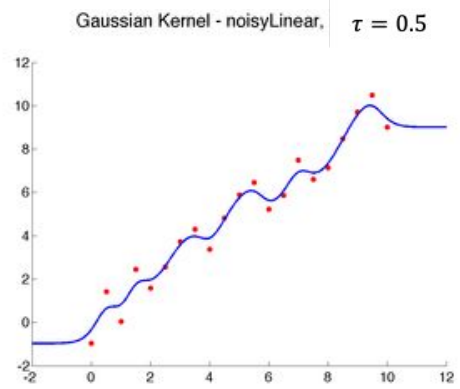
- Weighted average of training responses where weight is proportional to the similarity with the corresponding feature.

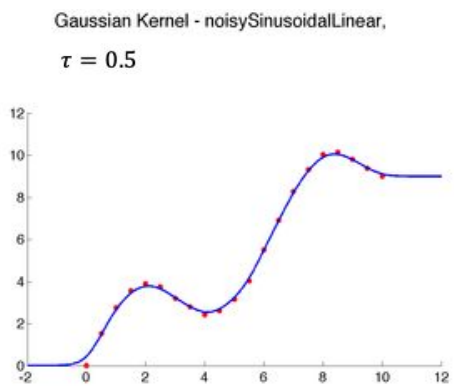
Kernel regression

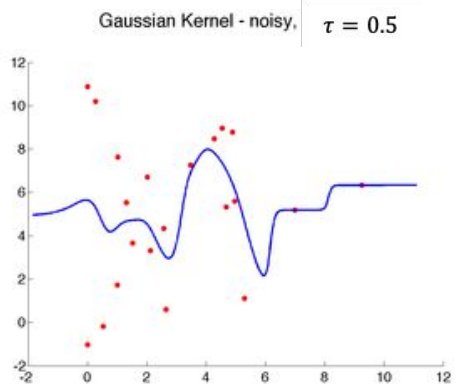
- Can use different kinds of kernels as they capture similarity between features differently.
  - Popular: Gaussian kernel with width τ:

$$\mathbf{k}(\mathbf{x}, \mathbf{x}') = \exp(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\tau^2})$$

- Examples







Kernels for classification

- We can just as easily use kernels for classification as well.
- Assume $y_i \in \{-1, +1\}$, return output as weighted majority:

$$h(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{N} \mathbf{k}(\mathbf{x}, \mathbf{x}_i) y_i\right)$$

- Compare it to k-nearest neighbor classification:

$$h(\mathbf{x}) = \text{sign}\left(\frac{1}{k} \sum_{(\mathbf{x}', y') \in kNN(\mathbf{x})}^{N} y_i\right)$$

Thank you!

Quiz: https://tinyurl.com/545kernels1

Next class: Support Vector Machines