EECS 545: Machine Learning

Lecture 8. Kernel methods

Honglak Lee

2/5/2025

UNIVERSITY OF MICHIGAN

Announcements

- HW2 due on Feb. 11th (Tuesday) at 11:55 PM

Outline

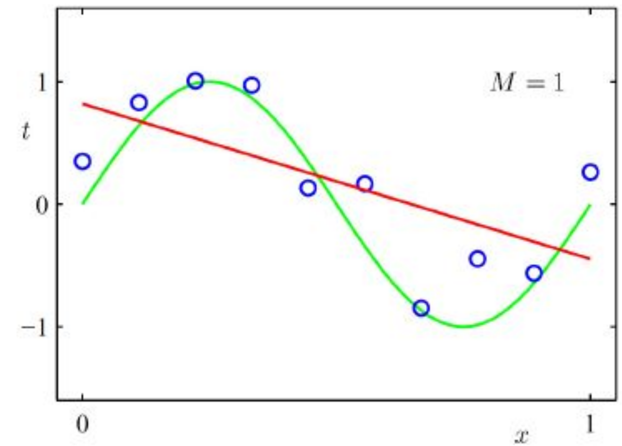
- Kernel methods: Motivation
- Kernel functions
- Kernel trick
  - Converting a ML model (objective function, prediction function, etc.) expressed with feature vectors to kernel functions
  - Kernel trick for linear regression
- Constructing (valid) kernels
- Kernel regression
  - Simple application of kernel method

Linear regression

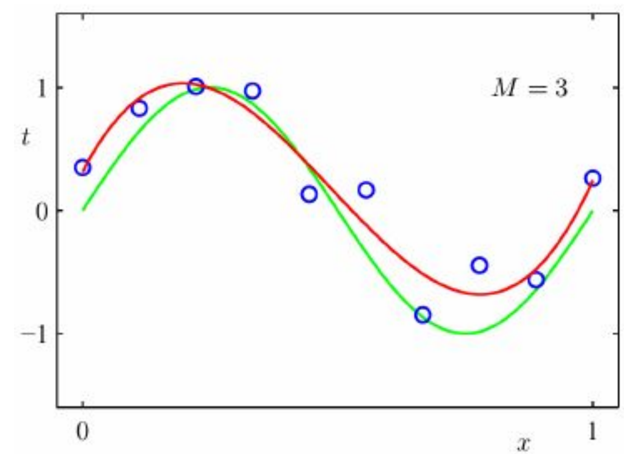
- Example: 1D regression
  - one input x, one output h(x)
- Linear model $h(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$ can only produce straight lines through origin
- Not very flexible/powerful
- How do we deal with this?

Feature mappings

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



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



Linear regression with (nonlinear) features

- Linear regression model

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

- Least-squares with L2 regression

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

- Closed form solution:

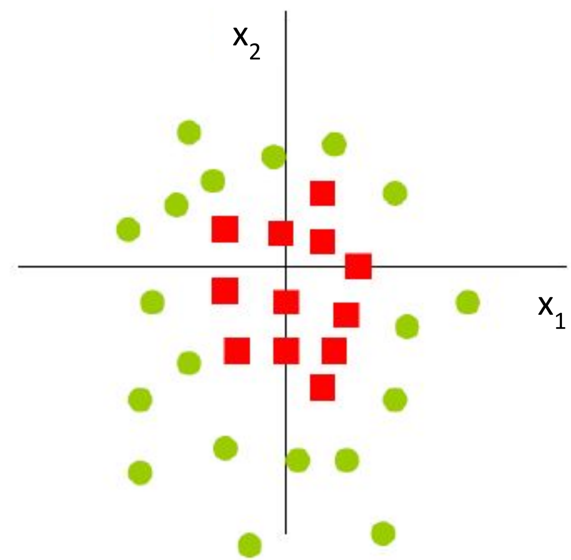
$$\mathbf{w} = (\Phi^\top \Phi + \lambda \mathbf{I})^{-1} \mathbf{y}$$

This is nice, but…

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

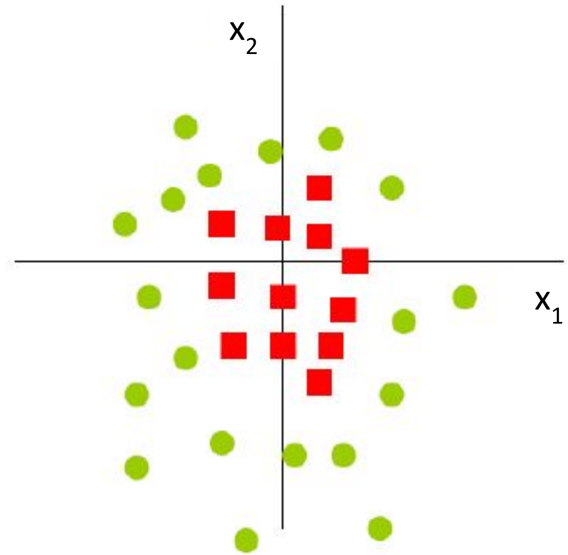
Linear classifiers

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



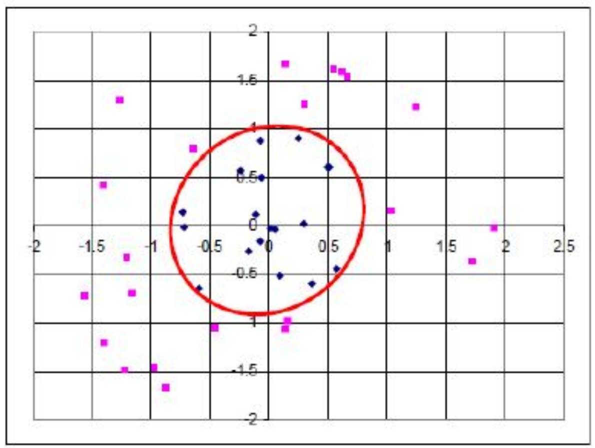
Linear classifiers with nonlinear features

- 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

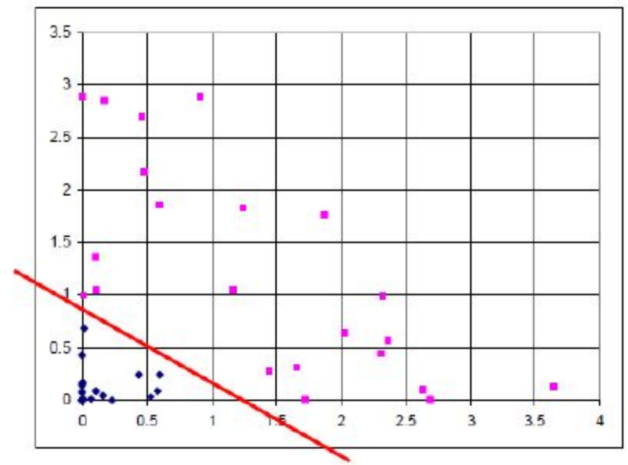


Linear classifiers with nonlinear features

- Another way: Replace $x_1 \to x_1^2, \quad x_2 \to x_2^2$



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 $\phi(\mathbf{x})$
  – The feature vector extracts important properties from $\mathbf{x}$.
  – 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 <u>inner products</u>
- Compute these inner products efficiently using a <u>kernel</u>

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: $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \phi(\mathbf{x}')$

- 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)

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

- Let's replace this by its square

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

- This is the same as inner product between

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

  – Or between

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

  – Note: solution is not unique.

Example: 2D input data

- Consider higher-order polynomial of degree p:

$$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^\top \mathbf{z})^p = \left(\sum_{j=1}^{M} x_j z_j \right)^p$$

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

- Feature mapping:

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

  – All monomials of degree *p*

Example: 2D input data

- Inhomogeneous polynomial up to degree p:

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

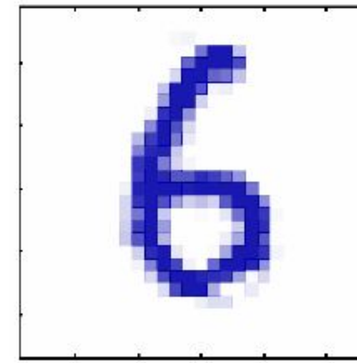
- Feature mapping:
  - All monomials of degree <= p

Example: handwritten digits images

- Take the pixel values and compute

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

  – 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.

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

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 $K = \Phi\Phi^\top$

$$K_{nm} = \phi(\mathbf{x}^{(n)})^\top \phi(\mathbf{x}^{(m)}) = k(\mathbf{x}^{(n)}, \mathbf{x}^{(m)})$$

- These represent the pairwise similarities among all the observed feature vectors.
  – Assumption: 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:

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

- 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

$$\phi_s = \frac{1}{N} \sum_{i=1}^{N} \phi(\mathbf{x}^{(i)})$$

Example: distance to the mean

- Mean of data points given by: $\phi_s = \frac{1}{N}\sum_{i=1}^{N}\phi(\mathbf{x}^{(i)})$
  - **cannot** be computed with kernel functions only

- Distance to mean:
  - **can** be computed with kernel functions only

$$\|\phi(\mathbf{x}) - \phi_s\|^2 = \langle\phi(\mathbf{x}), \phi(\mathbf{x})\rangle + \langle\phi_s, \phi_s\rangle - 2\langle\phi(\mathbf{x}), \phi_s\rangle$$

$$= \kappa(\mathbf{x}, \mathbf{x}) + \frac{1}{N^2}\sum_{i,j=1}^{N}\kappa(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) - \frac{2}{N}\sum_{i=1}^{N}\kappa(\mathbf{x}, \mathbf{x}^{(i)})$$

Kernel trick for linear regression

- Recall regression problems with error function

$$J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\{\mathbf{w}^\top \phi\left(x^{(n)}\right) - y^{(n)}\right\}^2 + \frac{\lambda}{2}\mathbf{w}^\top\mathbf{w}$$

- *J*(w) is minimized at

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

- 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 N x M 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

$$\Phi^\top \Phi$$ "covariance"

- Here, we will use the *N* x *N* *Gram* matrix

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

- Note that $K_{nm} = \phi(\mathbf{x}^{(n)})^\top \phi(\mathbf{x}^{(m)}) = k(\mathbf{x}^{(n)}, \mathbf{x}^{(m)})$
  - the pairwise similarities of all the data points in the training set
- Note that kernel methods use only K, not $\mathbf{\Phi}$

Kernel trick for linear regression

- Objective: $J(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\{\mathbf{w}^\top\phi\left(x^{(n)}\right) - y^{(n)}\right\}^2 + \frac{\lambda}{2}\mathbf{w}^\top\mathbf{w}$
- Another way to minimize *J*(w) is to set $\nabla_{\mathbf{w}} J(\mathbf{w}) = 0$

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \sum_{n=1}^{N}\{\mathbf{w}^\top\phi(x^{(n)}) - y^{(n)}\}\phi(x^{(n)}) + \lambda\mathbf{w} = 0$$

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

  – where $a_n = -\frac{1}{\lambda}\left\{\mathbf{w}^\top\phi\left(\mathbf{x}^{(n)}\right) - y^{(n)}\right\}$

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

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

Kernel trick for linear regression

- Objective function $J(\mathbf{w}) = \underbrace{\frac{1}{2}\sum_{n=1}^{N}\left\{\mathbf{w}^\top\phi\left(x^{(n)}\right) - y^{(n)}\right\}^2}_{\frac{1}{2}\mathbf{w}^\top\Phi^\top\Phi\mathbf{w}-\mathbf{w}^\top\Phi^\top\mathbf{y}+\frac{1}{2}\mathbf{y}^\top\mathbf{y}} + \frac{\lambda}{2}\mathbf{w}^\top\mathbf{w}$
- Substitute $\mathbf{w} = \Phi^\top\mathbf{a}$

$$
\begin{aligned}
J(\mathbf{a}) &= \frac{1}{2}\mathbf{a}^\top\Phi\Phi^\top\Phi\Phi^\top\mathbf{a} - \mathbf{a}^\top\Phi\Phi^\top\mathbf{y} + \frac{1}{2}\mathbf{y}^\top\mathbf{y} + \frac{\lambda}{2}\mathbf{a}^\top\Phi\Phi^\top\mathbf{a} \\
&= \frac{1}{2}\mathbf{a}^\top\mathbf{K}\mathbf{K}\mathbf{a} - \mathbf{a}^\top\mathbf{K}\mathbf{y} + \frac{1}{2}\mathbf{y}^\top\mathbf{y} + \frac{\lambda}{2}\mathbf{a}^\top\mathbf{K}\mathbf{a}
\end{aligned}
$$

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

- Setting the gradient w.r.t. *a* to zero:

$$
\begin{aligned}
\nabla_{\mathbf{a}}J(\mathbf{a}) = \mathbf{K}\mathbf{K}\mathbf{a} - \mathbf{K}\mathbf{y} + \lambda\mathbf{K}\mathbf{a} = 0 \\
(\mathbf{K} + \lambda\mathbf{I})\mathbf{a} = \mathbf{K}\mathbf{y}
\end{aligned}
$$

simplify (K is invertible)

- Solution (closed form):

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

Kernel trick for linear regression

- Objective function $J(\mathbf{w}) = \underbrace{\frac{1}{2}\sum_{n=1}^{N}\left\{\mathbf{w}^\top\phi\left(x^{(n)}\right) - y^{(n)}\right\}^2}_{\frac{1}{2}\mathbf{w}^\top\Phi^\top\Phi\mathbf{w}-\mathbf{w}^\top\Phi^\top\mathbf{y}+\frac{1}{2}\mathbf{y}^\top\mathbf{y}} + \frac{\lambda}{2}\mathbf{w}^\top\mathbf{w}$

- Substitute $\mathbf{w} = \Phi^\top\mathbf{a}$

$$
\begin{aligned}
J(\mathbf{a}) &= \frac{1}{2}\mathbf{a}^\top\Phi\Phi^\top\Phi\Phi^\top\mathbf{a} - \mathbf{a}^\top\Phi\Phi^\top\mathbf{y} + \frac{1}{2}\mathbf{y}^\top\mathbf{y} + \frac{\lambda}{2}\mathbf{a}^\top\Phi\Phi^\top\mathbf{a} \\
&= \frac{1}{2}\mathbf{a}^\top\mathbf{K}\mathbf{K}\mathbf{a} - \mathbf{a}^\top\mathbf{K}\mathbf{y} + \frac{1}{2}\mathbf{y}^\top\mathbf{y} + \frac{\lambda}{2}\mathbf{a}^\top\mathbf{K}\mathbf{a}
\end{aligned}
$$

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

- Solution (closed form): $\mathbf{a} = \left(\mathbf{K} + \lambda\mathbf{I}_N\right)^{-1}\mathbf{y}$

- Prediction for any arbitrary input *x*:

$$
\begin{aligned}
h(\mathbf{x}) &= \mathbf{w}^\top\phi(\mathbf{x}) = \phi(\mathbf{x})^\top\mathbf{w} = \phi(\mathbf{x})^\top\Phi^\top\mathbf{a} = \left(\Phi\phi(\mathbf{x})\right)^\top\mathbf{a} = \sum_{n=1}^{N} a_n k(\mathbf{x}^{(n)}, \mathbf{x}) \\
&= k(\mathbf{x})^\top\left(\mathbf{K} + \lambda\mathbf{I}_N\right)^{-1}\mathbf{y}
\end{aligned}
$$

definition:
$k(\mathbf{x}) = \left[k(\mathbf{x}^{(1)}, \mathbf{x}), \cdots, k(\mathbf{x}^{(N)}, \mathbf{x})\right]^\top$

Kernel trick for linear regression

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

  and the *Gram* matrix $\mathbf{K} = \Phi\Phi^\top$
- 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**):

$$h(\mathbf{x}) = \phi(\mathbf{x})^\top \mathbf{w} = \phi(\mathbf{x})^\top \Phi^\top \mathbf{a} = k(\mathbf{x})^\top (\mathbf{K} + \lambda \mathbf{I}_N)^{-1}\,\mathbf{y}$$

  – where $k(\mathbf{x}) = \left[k(\mathbf{x}^{(1)}, \mathbf{x}), \cdots, k(\mathbf{x}^{(N)}, \mathbf{x})\right]^\top$

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

Primal versus Dual

- Primal: $\mathbf{w} = \left(\Phi^\top \Phi + \lambda \mathbf{I}_M\right)^{-1} \Phi^\top \mathbf{y}$
- 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):
  1. Direct construction with feature vectors
  2. Mercer Theorem
  3. Composition of kernels with pre-defined rules

Constructing valid kernels: Method 1
Explicit Construction by defining feature vectors

- 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:

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \phi(\mathbf{x}') = \sum_{i=1}^{M} \phi_i(\mathbf{x}) \phi_i(\mathbf{x}')$$

Constructing valid kernels: Method 1
Explicit Construction by defining feature vectors

- Suppose we define a kernel function directly, such as

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

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

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

  – such that

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

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

Constructing valid kernels: Method 2
Mercer Theorem

- 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:

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

Constructing valid kernels: Method 3
Using Pre-Defined Rules

- 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 valid kernels: Method 3 Using Pre-Defined Rules

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

$$k(\mathbf{x}, \mathbf{x}') = ck_1(\mathbf{x}, \mathbf{x}') \tag{6.13}$$
$$k(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})k_1(\mathbf{x}, \mathbf{x}')f(\mathbf{x}') \tag{6.14}$$
$$k(\mathbf{x}, \mathbf{x}') = q\left(k_1(\mathbf{x}, \mathbf{x}')\right) \tag{6.15}$$
$$k(\mathbf{x}, \mathbf{x}') = \exp\left(k_1(\mathbf{x}, \mathbf{x}')\right) \tag{6.16}$$
$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') \tag{6.17}$$
$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}') \tag{6.18}$$
$$k(\mathbf{x}, \mathbf{x}') = k_3\left(\boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}')\right) \tag{6.19}$$
$$k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x}' \tag{6.20}$$
$$k(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x}_a, \mathbf{x}'_a) + k_b(\mathbf{x}_b, \mathbf{x}'_b) \tag{6.21}$$
$$k(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x}_a, \mathbf{x}'_a)k_b(\mathbf{x}_b, \mathbf{x}'_b) \tag{6.22}$$

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)

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

- Generalized Polynomial kernel - degree M

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

- 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)

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

Kernel regression

Kernel regression

- Recall k-nearest neighbor regression:

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

- Kernel regression:

$$h(\mathbf{x}) = \frac{\sum_i k(\mathbf{x}, \mathbf{x}^{(i)}) y^{(i)}}{\sum_j k(\mathbf{x}, \mathbf{x}^{(j)})} = \frac{1}{Z} \sum_i k(\mathbf{x}, \mathbf{x}^{(i)}) y^{(i)}$$

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

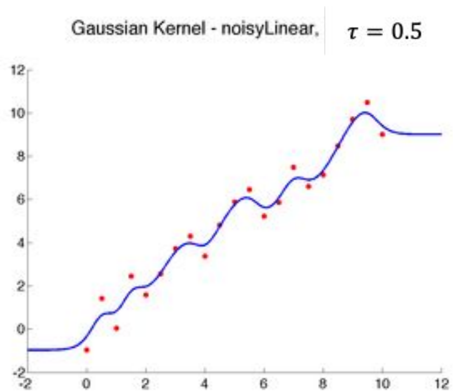
- Weighted average of training responses where weight is proportional to the similarity with the corresponding feature.

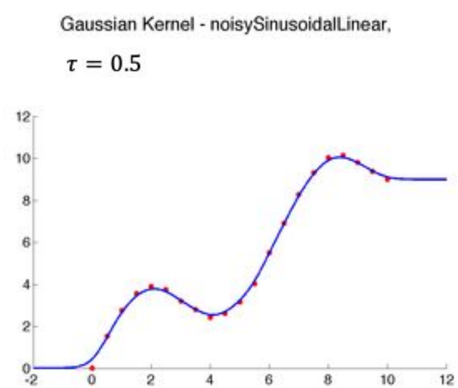
Kernel regression

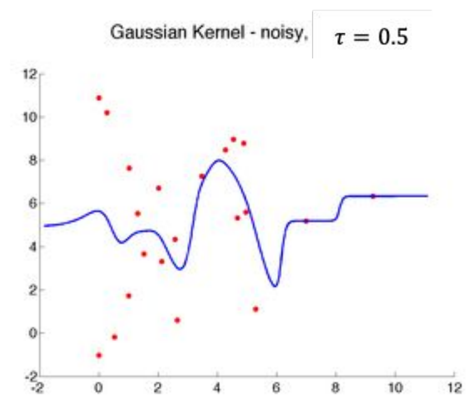
- Can use different kinds of kernels as they capture similarity between features differently.
  - Popular: Gaussian kernel with width τ:

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

- 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:

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

- Compare it to k-nearest neighbor classification:

$$h(\mathbf{x}) = \mathrm{sign}\left(\frac{1}{k} \sum_{(\mathbf{x}', y') \in \mathrm{KNN}(\mathbf{x})} y'\right)$$

Locally-weighted Linear Regression vs. Kernel regression

- Suppose we want to predict y given a query x.
- Locally-weighted linear regression
  - 1. Fit **w** to minimize $\sum_i r^{(i)}(y^{(i)} - \mathbf{w}^\top \phi(\mathbf{x}^{(i)}))^2 \leftarrow \mathbf{w} = (\Phi^\top R \Phi)^{-1} \Phi^\top R \mathbf{y}$
  - 2. Output $\mathbf{w}^\top \phi(\mathbf{x}^{(i)})$
  - Standard choice: $r^{(i)} = \exp\left(-\frac{\|\mathbf{x}^{(i)} - \mathbf{x}\|^2}{2\tau^2}\right)$

R is a diagonal matrix with
$R_{i,i} = \frac{1}{2} r^{(i)}$

τ: "kernel width"

- Kernel regression (Using Gaussian kernel)
  - output: $\frac{\sum_i \kappa(\mathbf{x}, \mathbf{x}^{(i)}) y^{(i)}}{\sum_i \kappa(\mathbf{x}, \mathbf{x}^{(i)})}$

    where $\kappa(\mathbf{x}, \mathbf{x}^{(i)}) = \exp(\frac{\|\mathbf{x}^{(i)} - \mathbf{x}\|^2}{2\tau})$

More generally, any distance metric (other than L2 or Euclidean distance) can be used.
Also, more general types of kernel function can be used.

Any feedback (about lecture, slide, homework, project, etc.)?

(via **anonymous** google form: https://forms.gle/fpYmiBtG9Me5qbP37)

Change Log of lecture slides:
https://docs.google.com/document/d/e/2PACX-1vSSIHJjkIypK7rKFSR1-5GYXyBCEW8UPtpSfCR9AR6M1l7K9ZQEmxfFwaWaW7kLDxusthsF8WlCyZJ-/pub