EECS 545: Machine Learning

Lecture 8. Kernel methods

Honglak Lee

2/7/2020

Outline

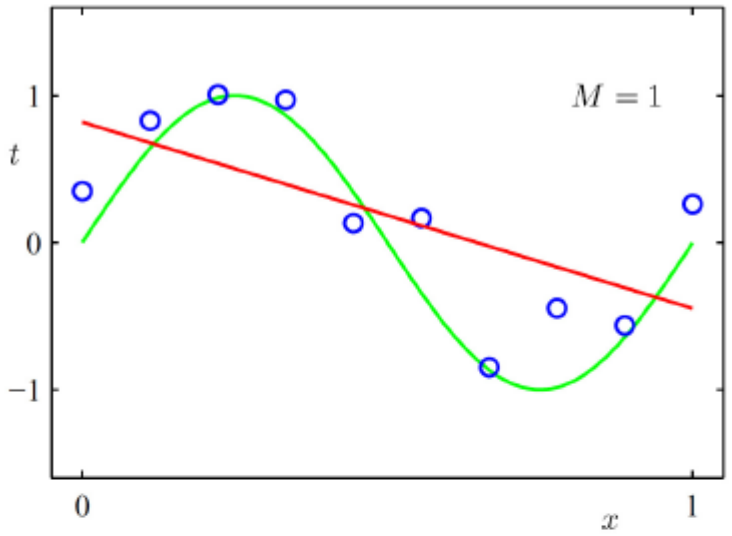
- Kernel methods: Motivation
- Kernel functions
- Kernel trick
- Dual representations
  - Example: linear regression
- Constructing kernels
- Kernel regression

Linear regression

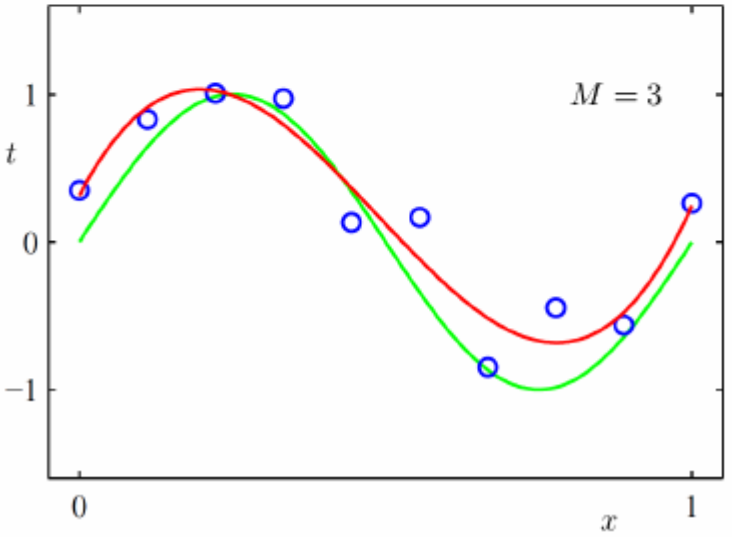
- Example: 1D regression
  - one input x, one output h(*x*)
- Linear model $h(x) = w^T x$ can only produce straight lines through origin
  - Not very flexible/powerful
- How do we deal with this?

Feature mappings

Replace x → (1,x) →



Replace x → $(1,x,x^2,x^3)$ →



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 regularization

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

- Closed form solution:

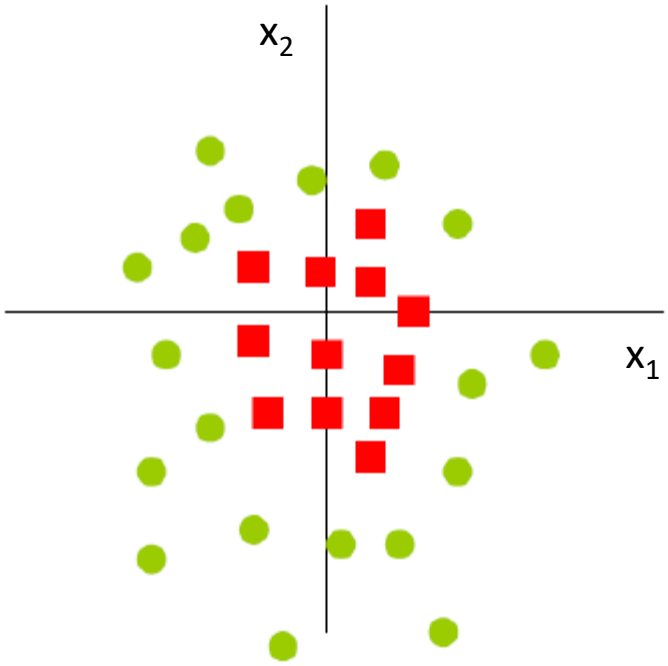
$$\mathbf{w} = (\Phi^T \Phi + \lambda \mathrm{I})^{-1} \Phi^T \boldsymbol{y}$$

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

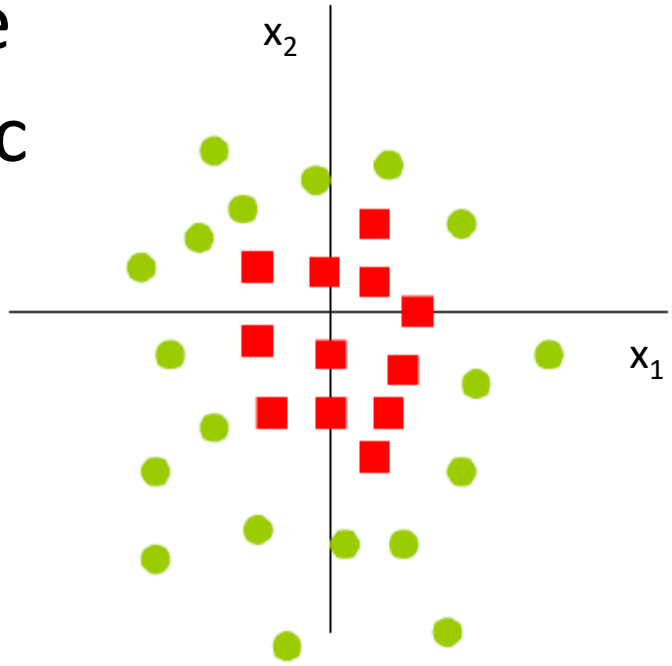
Linear Classifiers

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



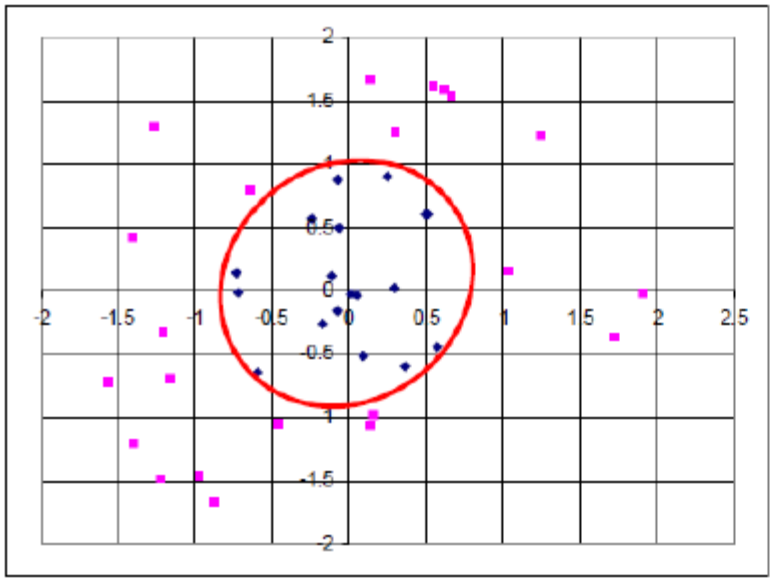
Linear classifiers with nonlinear features

- Add distance to origin $(x_1^2 + x_2^2)^{\frac{1}{2}}$ 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

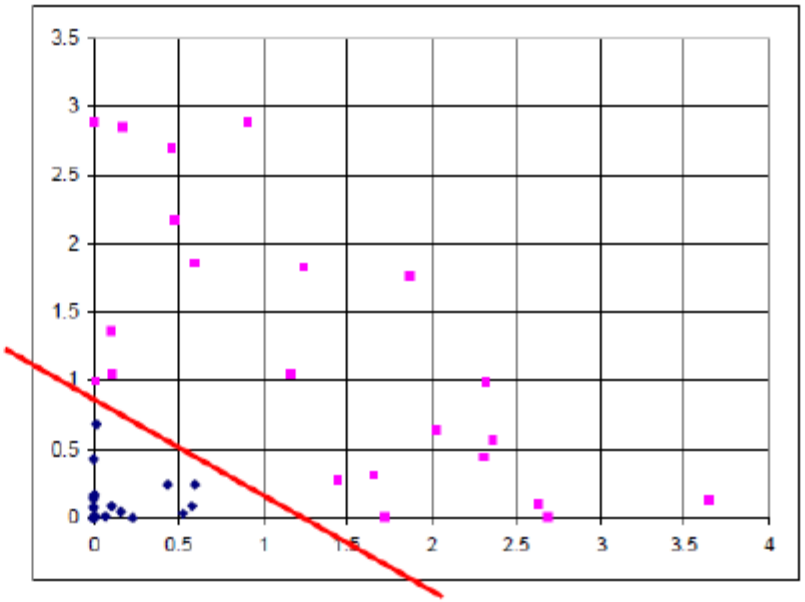


Linear Classifiers with nonlinear features

- 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 $\phi(\mathbf{x})$
  - 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*(**x**,**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*(**x**,**x'**) as one that can be expressed as an inner product, but we may not need to compute it that way.

Example: 2D input data

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

$$k(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T\mathbf{z} = x_1z_1 + x_2z_2$$

- Let's replace this by its square

$$k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T\mathbf{z})^2 = (x_1z_1 + x_2z_2)^2$$
$$= x_1^2z_1^2 + 2x_1z_1x_2z_2 + x_2^2z_2^2$$

- This is the same as inner product between $(x_1^2, \sqrt{2}x_1x_2, x_2^2)$ and $(z_1^2, \sqrt{2}z_1z_2, z_2^2)$
  - Or between $(x_1^2, x_1x_2, x_1x_2, x_2^2)$ and $(z_1^2, z_1z_2, z_1z_2, z_2^2)$.
  - Note: solution is not unique.

Example: 2D input data

- Consider higher-order polynomial of degree p:

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

$$= \sum_{(j_1,\ldots,j_M):\sum_j j_k = p} \binom{p}{j_1\, j_2\, \ldots j_M} (x_1 z_1)^{j_1} \ldots (x_M z_M)^{j_M}$$

- Feature mapping:

$$\phi(\mathbf{x}) = \left[\ldots, \binom{p}{j_1\, j_2\, \ldots j_M}^{\frac{1}{2}} (x_1)^{j_1} \ldots (x_M)^{j_M}, \ldots \right]$$

  - All monomials of degree p

Example: 2D input data

- Inhomogeneous polynomial up to degree p:

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

- Feature mapping:

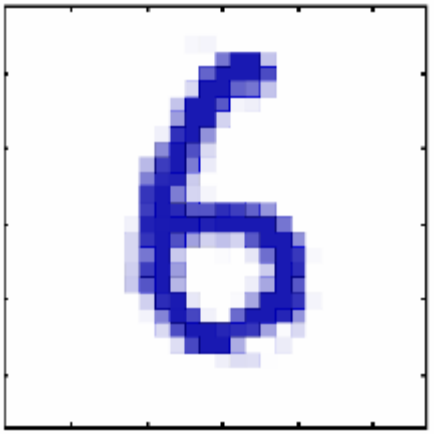
$$\phi(\mathbf{x}) = \text{all monomials of degree} \leq p.$$

Example: handwritten digits images

- Take the pixel values and compute

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

Here, **x**: 28x28 = 784 pixels



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

Kernel trick

- By using different definitions for inner product, we can compute inner products in a high dimensional space, with only the computational complexity of a low dimensional space
- Many algorithms can be expressed completely in terms of kernels $k(\mathbf{x},\mathbf{x}')$, rather than other operations on $\mathbf{x}$.
- In this case, you can replace one kernel with another, and get a new algorithm that works over a different domain.

Dual Representation and Kernel Trick

- The dual representation, and its solutions, are entirely written in terms of kernels.
- 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 can *not* 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:

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

- 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: $\boldsymbol{\phi}_S = \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{\phi}(\mathbf{x}_i)$

- Distance to mean:

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

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

Dual Representations 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

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

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

$$\mathbf{K} = \boldsymbol{\Phi}\boldsymbol{\Phi}^T$$ "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.

Dual Representations 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 parameter, instead of w.
- Transform *J*(w) to *J*(a) by substituting

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

Dual Representations for linear regression

- Objective function

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

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

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

Dual Representations 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**):

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

  – where

  $\mathbf{k}(\mathbf{x})$ has elements $k_n(\mathbf{x}) = k(\mathbf{x}_n, \mathbf{x})$

Primal versus Dual

- Primal: $\mathbf{w} = (\mathbf{\Phi}'\mathbf{\Phi} + \lambda \mathbf{I}_M)^{-1} \mathbf{\Phi}'\mathbf{y}$
- Dual: $\mathbf{a} = (\mathbf{K} + \lambda \mathbf{I}_N)^{-1} \mathbf{y}$
- 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 Kernels 1

- One can do *kernel engineering* to create kernels for particular purposes, expressing different kinds of similarity.

- Method 1: One way is to define the feature space mapping $\phi(\mathbf{x})$ and then define the inner product kernel

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

Constructing Kernels 1

- 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 Kernels 2

- A simpler way to test without having to construct $\phi(\mathbf{x})$
- 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:

$$a^T K a \equiv \sum_{ij} a_i K_{ij} a_j \geq 0, \forall a \in R^N$$

Constructing Kernels 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 3

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)

$$\kappa(\mathbf{x}, \mathbf{z}) = (\mathbf{x}'\mathbf{z})^2$$

- Generalized Polynomial kernel - degree M

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

- Gaussian Kernels

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

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

Locally-weighted Linear Regression vs. Kernel regression

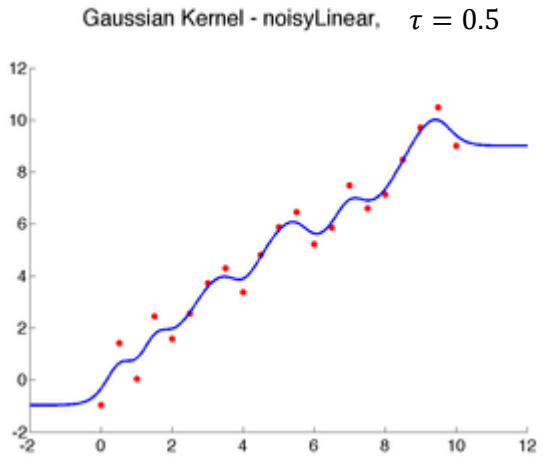
- Locally-weighted linear regression
  - 1. Fit $\mathbf{w}$ to minimize $\sum_i r^{(i)} \left(y^{(i)} - \mathbf{w}^T \phi\left(\mathbf{x}^{(i)}\right)\right)^2$
  - 2. Output $\mathbf{w}^T \phi\left(\mathbf{x}^{(i)}\right)$
  - Standard choice: $r^{(i)} = \exp\left(-\frac{\|\mathbf{x}^{(i)} - \mathbf{x}\|^2}{2\tau^2}\right)$ τ: "kernel width"
- Kernel regression (using Gaussian kernel)
  - output: $\frac{\sum_i K\left(\mathbf{x}, \mathbf{x}^{(i)}\right) y^{(i)}}{\sum_i K\left(\mathbf{x}, \mathbf{x}^{(i)}\right)}$

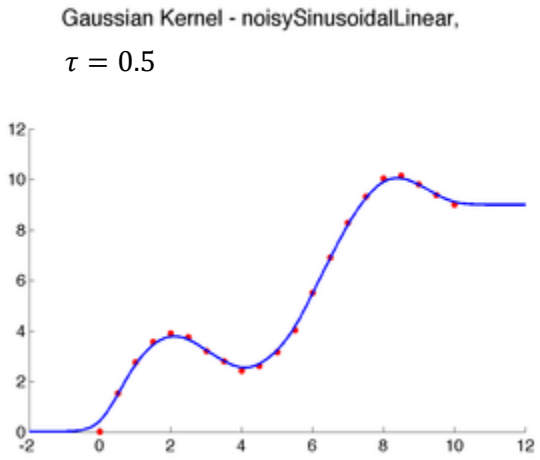
    where $K\left(\mathbf{x}, \mathbf{x}^{(i)}\right) = \exp\left(-\frac{\|\mathbf{x}^{(i)} - \mathbf{x}\|^2}{2\tau^2}\right)$

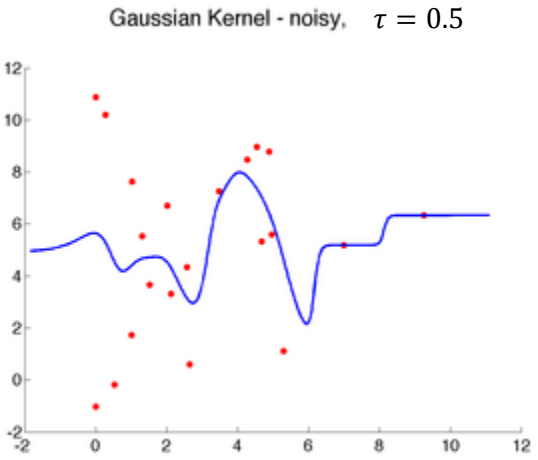
More generally, any distance metric (other than L2 or Eucleidian distance) can be used.
Also, more general types of kernel function can be used.

Kernel regression

- Examples







Kernel regression: Regression vs. Classification

- Note: it is very easy to formulate kernel regression into regression/classification
- Given training data $D = \{\mathbf{x}^{(i)}, y^{(i)}\}$, Kernel function $K(\cdot,\cdot)$ and input x
  - (regression) if $y \in \mathbf{R}$, return weighted average:

$$\frac{\sum_i K(\mathbf{x},\mathbf{x}^{(i)})y^{(i)}}{\sum_i K(\mathbf{x},\mathbf{x}^{(i)})}$$

  - (classification) if $y \in \pm 1$, return weighted majority:

$$\text{sign}\left(\sum_i K(\mathbf{x},\mathbf{x}^{(i)})y^{(i)}\right)$$

Locally weighted regression vs. kernel regression

- Both methods are “instance-based learning”.
  - Only observations (training set) close to the query point are considered (highly weighted) for regression computation.
  - Kernel determines how to assign weights to training examples (similarity to the query point x)
    - Free to choose types of kernels
  - Both can suffer when the input dimension is high.
- Difference:

Locally weighted regression	Kernel regression
Weighted regression	Weighted mean
Slow, but more accurate	Faster, but less accurate

Link: https://bit.ly/2vfEEb6

Quiz

1. (True/False) Using the kernel trick, one can get non-linear decision boundaries using algorithms designed originally for linear models.

2. (True/False) Logistic regression cannot be kernelized.

Link: https://bit.ly/2vfEEb6

Quiz

3. (True/False) Linear classifier with quadratic kernel $\phi(\boldsymbol{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$ can correctly classify the training set for XOR. [Hint: we allow for pre-processing the data, such as centering which makes each coordinate mean-zero and unit variance.]

4. What is the purpose of the Kernel Trick?

a) To transform the problem from regression to classification
b) To transform the data from nonlinearly separable to linearly separable
c) To transform the problem from supervised to unsupervised learning.