$\LaTeX$ command declarations here.

EECS 545: Machine Learning
Lecture 09: Kernel Methods

- Instructor: **Jacob Abernethy**
- Date: February 8, 2016

Lecture Exposition Credit: Benjamin Bray, Saket Dewangan

Outline

- Feature Mapping
  - Regression
  - Classification
- Kernel Methods
  - Kernel functions
  - Kernel trick
  - Constructing kernels
- Dual Representation using Kernels
  - Linear Regression
- Kernel Regression

Reading List

- Required:
  - **[PRML]**, §6.1: Dual Representation
  - **[PRML]**, §6.2: Constructing Kernels
  - **[PRML]**, §6.3: Radial Basis Function Networks
- Optional:
  - **[MLAPP]**, §14.2: Kernel Functions
  - **Everything You Wanted to Know about the Kernel Trick (But were too Afraid to Ask)** (http://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html)

> In this lecture, we will cover kernel methods. For feature mapping $\phi(\mathbf{x})$, we will see mapping features to higher dimensional space is advantageous in machine learning, but higher dimension brings greater computational complexity. The kernel trick, our protagonist, comes to the rescue, with the ability to implicitly handle higher dimensional features at low cost. We will see how kernel functions are defined, how to construct kernels and how to use the kernel trick in machine learning. Kernels give us a dual representation of learning algorithms which entirely written in terms of kernel functions. Particularly, we will show how to transform regularized linear regression into a dual representation. Finally we review cover kernel regression, a technique similar to locally weighted least squares.

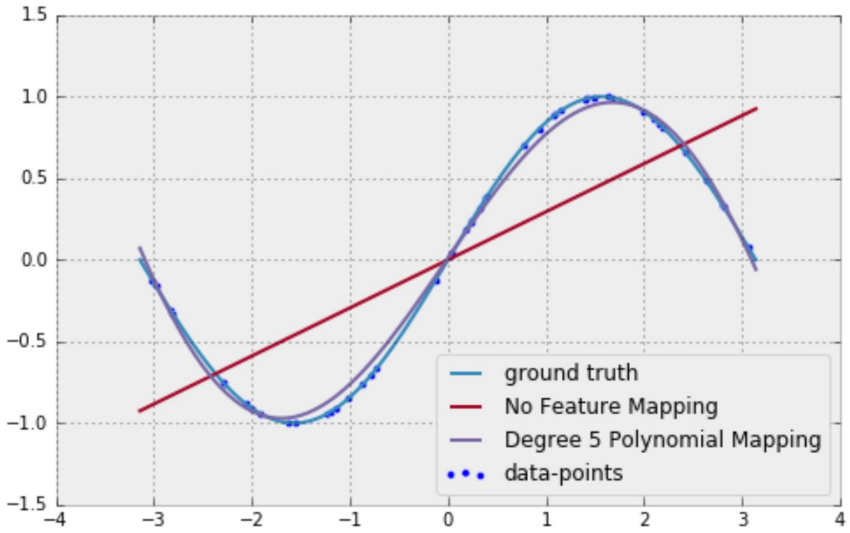
Feature Mapping

Review: Feature Mapping

- **Feature mapping** $\phi(\mathbf{x}) : \mathcal{X} \mapsto \mathbb{R}^M$ maps data into a feature space.
  - In general, features are nonlinear functions of the data
    - Recall we use polynomial features in linear regression
  - It could be that $\phi(\mathbf{x}) = \mathbf{x}$, i.e. no mapping exists
  - Each feature $\phi_j(\mathbf{x})$ extracts important properties from x
  - Feature mapping could boost the performance of learning algorithms.

Feature Mapping: Linear Regression

- Without feature mapping, linear model $y(\mathbf{x}) = w^T\mathbf{x}$ can only produce straight lines through origin.
  - Not very flexible / powerful and can't handle nonlinearities
- We could replace $\mathbf{x}$ by $\phi(\mathbf{x}) = (\mathbf{x}, \mathbf{x}^2, \cdots, \mathbf{x}^p)$, so $y(\mathbf{x}) = w^T\phi(\mathbf{x})$
  - An appropriate feature mapping could generate nice regressor, eg. p=5 in the following plots
- Note that in this lecture, notation $(x_1, x_2, \cdots, x_N)$ denote a column vector.



Feature Mapping: Linear Classification

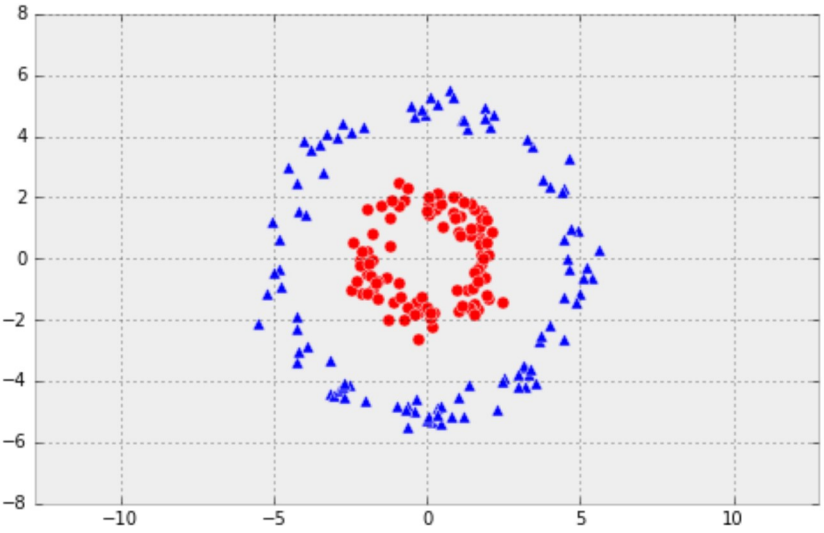
- **Recall:** Linear (binary) classifiers separate data with a **hyperplane** in feature space,

$$h(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w}^T\phi(\mathbf{x}) > 0 \\ 0 & \text{otherwise} \end{cases}$$

- **Examples:** Differ only in how the weights w are learned.
  - Logistic regression, LDA(GDA with shared covariance), Fisher's Linear Discriminant, Perceptron (Lec 07 & 08).

Feature Mapping: Linear Classification

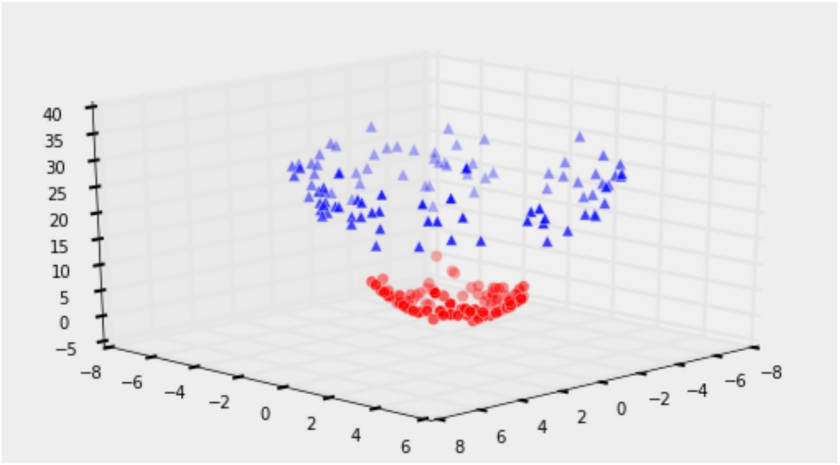
- **Problem:** The following data is **NOT** linearly separable if we want to learn a linear classifier with form $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$

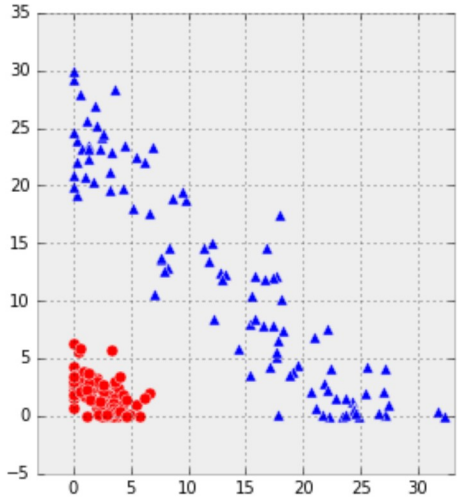


- $\mathbf{x} = (x_1, x_2)$, so $\mathbf{w}^T\mathbf{x} = w_1x_1 + w_2x_2$ which is a straight line through the origin.
- Obviously, no linear classifier can separate the blue rectangles from the red circles.

Feature Mapping: Linear Classification

- **Solution 1 (Left):** Add squared-distance-to-origin $(x_1^2 + x_2^2)$ as third feature. So $\phi(\mathbf{x}) = (x_1, x_2, x_1^2 + x_2^2)$
- **Solution 2 (Right):** Use features $\phi(\mathbf{x}) = (x_1^2, x_2^2)$





- Linear **separable** now with the new features!

Feature Mapping: Advantages

- From examples of regression and classification, we see
  - Data has been mapped via $\phi(\mathbf{x})$ to a new, higher-dimensional (possibly infinite!) space
  - Certain *mapping* are better for certain problems

Feature Mapping: Disadvantages

- High feature dimension M brings high computational complexity
  - Recall in regularized linear regression, we have solution
    $$\mathbf{w} = (\Phi^T\Phi + \lambda I)^{-1}\Phi^T\mathbf{t}$$
    Design matrix $\Phi \in \mathbb{R}N \times M$ of which N is number of samples and M is number features
  - $\Phi^T\Phi \in \mathbb{R}^{M\times M}$
  - Inversion $(\Phi^T\Phi + \lambda I)^{-1}$ has complexity $O(M^3)$!
- We can never handle features with *infinite* dimension
- **Kernel** methods come to the rescue
  - Implicitly handle high dimensional features with low computational complexity
  - Even possible to handle *infinite* dimensional features (we will see later)

Kernel Methods

> Uses content from **[PRML]** and Wikipedia, "Kernel Method" (https://en.wikipedia.org/wiki/Kernel_method)

Kernel Methods: Intro

- Many algorithms depend on the data only through pairwise **inner products** between data points,
  $$\langle \mathbf{x}_1, \mathbf{x}_2 \rangle = \mathbf{x}_2^T\mathbf{x}_1$$
  - Inner product represents *similarity* between data points.
- Inner products can be replaced by **Kernel Functions**, capturing more general notions of similarity.
  - No longer need coordinates!

Kernel Functions: Definition

- A **kernel function** $\kappa(\mathbf{x}, \mathbf{x}')$ is intended to measure the similarity between $\mathbf{x}$ and $\mathbf{x}'$.
  - Eg. $\mathbf{x}^T\mathbf{x}'$, $(\mathbf{x}^T\mathbf{x}')^p$, $\|\mathbf{x} - \mathbf{x}'\|^2$, $\exp\left(\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2\sigma^2}\right)$
- In general, a **valid** kernel function $\kappa(\mathbf{x}, \mathbf{x}')$ satisfies that for every set $\mathbf{x}_1, \ldots, \mathbf{x}_N$, the matrix
  $$\begin{bmatrix} \kappa(\mathbf{x}_1,\mathbf{x}_1) & \kappa(\mathbf{x}_1,\mathbf{x}_2) & \cdots & \kappa(\mathbf{x}_1,\mathbf{x}_N) \\ \kappa(\mathbf{x}_2,\mathbf{x}_1) & \kappa(\mathbf{x}_2,\mathbf{x}_2) & \cdots & \kappa(\mathbf{x}_2,\mathbf{x}_N) \\ \vdots & \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}_N,\mathbf{x}_1) & \kappa(\mathbf{x}_N,\mathbf{x}_2) & \cdots & \kappa(\mathbf{x}_N,\mathbf{x}_N) \end{bmatrix}$$
  is a **positive-semidefinite** (PSD) matrix.

Kernel Functions: Kernel Function $\leftrightarrow$ Implicit Feature Mapping

- For every **valid** kernel function $\kappa(\mathbf{x}, \mathbf{x}')$, there exists an implicit feature mapping $\phi(x)$ such that
$$\kappa(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$$
  - Kernel function corresponds to inner product of features in some high-dimensional space
  - We will show this with some examples
- The converse is also **true**. Given some feature mapping $\phi(\mathbf{x})$, then $\kappa(\mathbf{x}, \mathbf{x}')$ defined by
$$\kappa(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$$
  is a valid kernel function
- We could prove a kernel is valid by finding an implicit feature mapping!
- The *incredible* result above follows from **Mercer's Theorem**,
  - Generalizes the fact that *every positive-definite matrix corresponds to an inner product*
  - For more info, see Hsing & Eubank 2015, "Theoretical Foundations of Functional Data Analysis" (http://www.wiley.com/WileyCDA/WileyTitle/productCd-0470016914.html)

Kernel Functions: Example—Quadratic Polynomial Kernel

- **Kernel:** For $\mathbf{x} = (x_1, x_2)$ and $\mathbf{x}' = (x_1', x_2')$, define
$$\begin{aligned}\kappa(\mathbf{x}, \mathbf{x}') &= (\mathbf{x}^T \mathbf{x}')^2 \\ &= (x_1 x_1' + x_2 x_2')^2 \\ &= x_1^2 x_1'^2 + 2 x_1 x_2 x_1' x_2' + x_2^2 x_2'^2\end{aligned}$$
- **Mapping:** Equivalent to the standard inner product when either
$$\phi(\mathbf{x}) = (x_1^2, \sqrt{2} x_1 x_2, x_2^2)$$
$$\phi(\mathbf{x}) = (x_1^2, x_1 x_2, x_1 x_2, x_2^2)$$
- Implicit mapping is **not** unique!
- Note $\mathbf{x}$ and $\mathbf{x}'$ denote two data points. $\mathbf{x}'$ does **NOT** denote transpose of x here.

Kernel Functions: Example—Polynomial Kernel

- **Kernel:** Higher-order polynomial of degree p,
$$\kappa(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}')^p = \left(\sum_{k=1}^M x_k x_k' \right)^p$$
- **Mapping:** Implicit feature vector $\phi(\mathbf{x})$ contains all monomials of degree p
- **Example:** For $\mathbf{x} = (x_1, x_2)$, $\mathbf{x}' = (x_1', x_2')$
  - When $p = 2$, as previous example, we have
$$\phi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2} x_1 x_2)$$
  - When $p = 3$, we have
$$\phi(\mathbf{x}) = (x_1^3, x_2^3, \sqrt{3} x_1^2 x_2, \sqrt{3} x_1 x_2^2)$$
- Try to verify this!

Kernel Functions: Example—Generalized Polynomial Kernel

- **Kernel:** Inhomogeneous polynomial up to degree p, for $c > 0$,
$$\kappa(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}' + c)^p = \left(c + \sum_{k=1}^M \mathbf{x}_k \mathbf{x}_k' \right)^p$$
- **Mapping:** Implicit feature vector $\phi(\mathbf{x})$ contains all monomials of degree $\leq p$
- **Example:** For $\mathbf{x} = (x_1, x_2)$, $\mathbf{x}' = (x_1', x_2')$
  - When $p = 2$, we have
$$\phi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2} x_1 x_2, \sqrt{2} \sqrt{c} x_1, \sqrt{2} \sqrt{c} x_2, c)$$
  - When $p = 3$, we have
$$\phi(\mathbf{x}) = (x_1^3, x_2^3, \sqrt{c}^3, \sqrt{3} x_1^2 x_2, \sqrt{3} x_1^2 \sqrt{c}, \sqrt{3} x_2^2 x_1, \sqrt{3} x_2^2 \sqrt{c}, \sqrt{3} c x_1, \sqrt{3} c x_2, \sqrt{6} x_1 x_2 \sqrt{c})$$
  - Note that we could view $\kappa(\mathbf{x}, \mathbf{x}')$ as $(\tilde{\mathbf{x}}^T \tilde{\mathbf{x}}')^p$ of which $\tilde{\mathbf{x}} = (\mathbf{x}, \sqrt{c})$ and $\tilde{\mathbf{x}}' = (\mathbf{x}', \sqrt{c})$
- Try to verify this!

Kernel Functions: Example—Gaussian Kernel

- **Kernel:**

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

- **Mapping:** Implicit feature vector $\phi(\mathbf{x})$ has **infinite** dimension!
- **Not related to Gaussian pdf!**
- Translation invariant (depends only on distance between points)
- Derivation of infinite dimension feature is in the notes

Remark

- Derivation of infinite features of Gaussian Kernel
  - Expand the Gaussian kernel:

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2\sigma^2}\right) = \exp\left(-\frac{\|\mathbf{x}\|^2}{2\sigma^2}\right)\exp\left(\frac{\mathbf{x}^T\mathbf{x}'}{\sigma^2}\right)\exp\left(-\frac{\|\mathbf{x}'\|^2}{2\sigma^2}\right)$$

    of which

$$\exp\left(\frac{\mathbf{x}^T\mathbf{x}'}{\sigma^2}\right) = \sum_{n=0}^{\infty}\frac{1}{n!}\left(\frac{\mathbf{x}^T\mathbf{x}'}{\sigma^2}\right)^n = \sum_{n=0}^{\infty}\frac{1}{n!\sigma^{2n}}\left(\mathbf{x}^T\mathbf{x}'\right)^n$$

  - We already know $(\mathbf{x}^T\mathbf{x}')^n$ is a valid kernel. So let $\psi_n(\mathbf{x})$ denote the feature mapping of $(\mathbf{x}^T\mathbf{x}')^n$, i.e.

$$\psi_n(\mathbf{x})^T\psi_n(\mathbf{x}) = (\mathbf{x}^T\mathbf{x}')^n$$

    Then we have

$$\exp\left(\frac{\mathbf{x}^T\mathbf{x}'}{\sigma^2}\right) = \sum_{n=0}^{\infty}\frac{1}{n!\sigma^{2n}}\psi_n(\mathbf{x})^T\psi_n(\mathbf{x})$$

$$= \left(\sqrt{\frac{1}{0!}}\psi_0(\mathbf{x}), \sqrt{\frac{1}{1!\sigma^2}}\psi_1(\mathbf{x}), \ldots, \sqrt{\frac{1}{n!\sigma^{2n}}}\psi_n(\mathbf{x})\right)^T\left(\sqrt{\frac{1}{0!}}\psi_0(\mathbf{x}'), \sqrt{\frac{1}{1!\sigma^2}}\psi_1(\mathbf{x}'), \ldots, \sqrt{\frac{1}{n!\sigma^{2n}}}\psi_n(\right.$$

$$\triangleq \eta(\mathbf{x})^T\eta(\mathbf{x}')$$

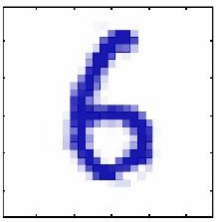
    of which $\eta(\mathbf{x})$ is a new feature mapping with **infinite** dimension.
  - So

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x}\|^2}{2\sigma^2}\right)\eta(\mathbf{x})^T\eta(\mathbf{x}')\exp\left(-\frac{\|\mathbf{x}'\|^2}{2\sigma^2}\right) \triangleq \phi(\mathbf{x})^T\phi(\mathbf{x}')$$

    of which $\phi(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x}\|^2}{2\sigma^2}\right)\eta(\mathbf{x})$ is the final infinite dimension feature mapping of $\kappa(\mathbf{x}, \mathbf{x}')$

Kernel Functions: Handwritten Example

- For 28×28 handwritten image below



- Take the pixel values and compute $\kappa(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T\mathbf{x}' + 1)^p$
  - here, $\mathbf{x}$ is the vectorized image, which has $28 \times 28 = 784$ pixels
- According to the generalized polynomial kernel, for $\dim(\mathbf{x}) = 784$ and $p = 4$, the dimension of features $\dim(\phi(x)) \approx 1.6 \times 10^{10}$!
- **Recall**, high-dimensional feature could boost the performance of machine learning algorithm
  - Kernel Functions will be the bridge!
- But before we apply kernel functions to ML algorithms, let's first see how to construct kernels.

Remark

- How is the 1.6×10^{10} computed?
  - Define $\tilde{\mathbf{x}} = (\mathbf{x}, 1) \in \mathbb{R}^{785}$ and $\tilde{\mathbf{x}}' = (\mathbf{x}', 1) \in \mathbb{R}^{785}$, so we have

$$\begin{aligned}\kappa(\mathbf{x}, \mathbf{x}') &= (\tilde{\mathbf{x}}^T \tilde{\mathbf{x}}')^4 = (\sum_{i=1}^{785} \tilde{x}_i \tilde{x_i}')^4 \\ &= (\tilde{x}_1 \tilde{x}'_1 + \tilde{x}_2 \tilde{x}'_2 + \cdots + \tilde{x}_{785} \tilde{x}'_{785})^4 \\ &\triangleq (y_1 + y_2 + \cdots + y_{785})^4\end{aligned}$$

  - The dimension equals the number of monomials with degree 4.
  - It's just a combinatorial problem. Now lets count.
  - In the table, m, n, p, q are *distinct* and $m, n, p, q = 1, 2, \ldots, 785$.

Monomial Combination	y_m^4	$y_m^3 y_n$	$y_m^2 y_n^2$	$y_m^2 y_n y_p$	$y_m y_n y_p y_q$
Counting	$\binom{785}{1}$	$\binom{785}{2}\binom{3}{1}$	$\binom{785}{2}$	$\binom{785}{3}\binom{3}{1}$	$\binom{785}{4}$

- Add them up, we could get 1.59×10^{10}

Kernel Functions: Constructing Kernels

- **Method 1:** Explicitly define a feature space mapping $\phi(\mathbf{x})$ and use **inner product kernel**

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

- **Method 2:** Explicitly define a kernel $\kappa(\mathbf{x}, \mathbf{x}')$ and identify the implicit feature map, e.g.

$$\kappa(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^T \mathbf{x}')^2 \quad \Longrightarrow \quad \phi(\mathbf{x}) = (\mathbf{x}_1^2, \sqrt{2}\mathbf{x}_1\mathbf{x}_2, \mathbf{x}_2^2)$$

  - Kernels help us avoid the complexity of explicit feature mappings.
- **Method 3:** Define a similarity function $\kappa(x, x')$ and use **Mercer's Condition** to verify that an implicit feature map *exists* without finding it.
  - Define the Gram matrix K to have elements $K_{nm} = \kappa(\mathbf{x}_n, \mathbf{x}_m)$
  - Then K must be **positive semidefinite** for all possible choices of the data set $\{x_n\}$, i.e.

$$a^T K a \equiv \sum_i \sum_j a_i K_{ij} a_j \geq 0 \quad \forall\, a \in \mathbb{R}^n$$

Kernel Functions: Building Blocks

- Given valid kernels $\kappa_1(\mathbf{x}, \mathbf{x}')$ and $\kappa_2(\mathbf{x}, \mathbf{x}')$, the following new kernels will also be valid:

$\kappa(\mathbf{x}, \mathbf{x}') = c\kappa_1(\mathbf{x}, \mathbf{x}')$	Constant $c > 0$
$\kappa(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})\kappa_1(\mathbf{x}, \mathbf{x}')f(\mathbf{x}')$	$\forall$ function $f(\cdot)$
$\kappa(\mathbf{x}, \mathbf{x}') = q(\kappa_1(\mathbf{x}, \mathbf{x}'))$	$\forall$ polynomial $q(\cdot)$ with nonnegative coeffs
$\kappa(\mathbf{x}, \mathbf{x}') = \exp(\kappa_1(\mathbf{x}, \mathbf{x}'))$	
$\kappa(\mathbf{x}, \mathbf{x}') = \kappa_1(\mathbf{x}, \mathbf{x}') + \kappa_2(\mathbf{x}, \mathbf{x}')$	
$\kappa(\mathbf{x}, \mathbf{x}') = \kappa_1(\mathbf{x}, \mathbf{x}')\kappa_2(\mathbf{x}, \mathbf{x}')$	
$\kappa(\mathbf{x}, \mathbf{x}') = \kappa_1(\phi(\mathbf{x}), \phi(\mathbf{x}'))$	$\phi(\mathbf{x})$ is any feature mapping
$\kappa(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T A \mathbf{x}$	A is any symmetric positive semidefinite matrix

- Popular Kernel Functions

$$\begin{aligned}\kappa(\mathbf{x}, \mathbf{x}') &= (\mathbf{x}^T \mathbf{x}')^p \\ \kappa(\mathbf{x}, \mathbf{x}') &= (\mathbf{x}^T \mathbf{x}' + c)^p && c > 0 \\ \kappa(\mathbf{x}, \mathbf{x}') &= \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right)\end{aligned}$$

Kernel Methods: Kernel Trick

- So far, we have seen
  - kernel is related to high-dimension features
  - how to construct features.
- **But**, how to apply kernels to ML algorithm?
- **Idea:** if an algorithm is formulated in such a way that the $\mathbf{x}$ enters only through inner product, then we can replace the inner product with some other choice of kernel.
  - i.e. $\mathbf{x}^T\mathbf{x}'$ or $\phi(\mathbf{x})^T\phi(\mathbf{x}') \rightarrow \kappa(\mathbf{x}, \mathbf{x}')$
  - This is called **kernel trick** or **kernel substitution**
  - To use the kernel trick, we must formulate 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
- By using different definitions for inner product, we can manipulate features in high dimensional space with only the computational complexity of a low dimensional space.

Kernel Trick: Example—Distance

- Distance between samples can be expressed in inner products:

$$\begin{aligned}||\mathbf{x} - \mathbf{x}'||^2 &= \langle \mathbf{x} - \mathbf{x}', \mathbf{x} - \mathbf{x}' \rangle \\ &= \langle \mathbf{x}, \mathbf{x} \rangle - 2\langle \mathbf{x}, \mathbf{x}' \rangle + \langle \mathbf{x}, \mathbf{x}' \rangle\end{aligned}$$

  After kernel substitution, we have

$$||\mathbf{x} - \mathbf{x}'||^2 = \kappa(\mathbf{x}, \mathbf{x}) - 2\kappa(\mathbf{x}, vx') + \kappa(\mathbf{x}', \mathbf{x}')$$

- With kernels, we are no longer limited to distance defined by ℓ_2 norm.
- This generalized distance can be used in distance-based algorithm, eg. **k-nearest neighbor searches**.

Kernel Trick: Example—Distance to the Mean

- Mean of data points given by: $\bar{\mathbf{x}} = \frac{1}{N}\sum_{i=1}^N \mathbf{x}_i$
- Distance to the mean:

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

Kernel Trick: Question

Question: Can you determine the **mean** of data in the mapped feature space through kernel operations only?

Answer: No, you cannot compute any point explicitly.

Dual Representations: Linear Regression

> Following **[PRML]** Chapter 6.1

Dual Representations: Linear Regression

- In this part, we will transform linear regression into its dual representation
  - i.e. express the algorithm entirely with kernels
  - Same as kernel substitution, we will replace inner product with kernels.
- **Review**
  - The solution to regularized linear regression is

$$\mathbf{w} = \left(\Phi^T\Phi + \lambda I_M\right)^{-1}\Phi^T\mathbf{t}$$

of which

$$\Phi = \begin{bmatrix} - & \phi(\mathbf{x}_1)^T & - \\ & \vdots & \\ - & \phi(\mathbf{x}_N)^T & - \end{bmatrix}_{N\times M} \qquad I_M = \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{bmatrix}_{M\times M} \qquad \mathbf{t} = \begin{bmatrix} t_1 \\ \vdots \\ t_N \end{bmatrix}_{N\times 1}$$

Recall Φ is **design** matrix. N is number of data points. M is dimension of features.

  - The prediction is $y(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x})$
- **Preliminary**
  - Matrix Inversion Lemma (We will use twice later!)

$$(A+UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1}+VA^{-1}U)^{-1}VA^{-1}$$

Dual Representations: Derivation

- Using matrix inversion lemma

$$(A+UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1}+VA^{-1}U)^{-1}VA^{-1}$$

to expand the solution:

$$\begin{aligned}
\mathbf{w} &= \left(\lambda I_M + \Phi^T I_N \Phi\right)^{-1}\Phi^T\mathbf{t} \\
&= \left[\lambda^{-1}I_M - \lambda^{-1}I_M\Phi^T\left(I_N + \Phi\lambda^{-1}I_M\Phi^T\right)^{-1}\Phi\lambda^{-1}I_M\right]\Phi^T\mathbf{t} \\
&= \left[\lambda^{-1}\Phi^T - \lambda^{-1}\Phi^T\left(\lambda I_N + \Phi\Phi^T\right)^{-1}\Phi\Phi^T\right]\mathbf{t} \\
&= \Phi^T\left[\lambda^{-1}I_N - \lambda^{-1}\left(\lambda\left(\Phi\Phi^T\right)^{-1} + I_N\right)^{-1}\right]\mathbf{t} \\
&= \Phi^T\left[\lambda^{-1}I_N - \lambda^{-2}\left(\left(\Phi\Phi^T\right)^{-1} + \lambda^{-1}I_N\right)^{-1}\right]\mathbf{t} \\
&= \Phi^T\left[(\lambda I_N)^{-1} - (\lambda I_N)^{-1}I_N\left(\left(\Phi\Phi^T\right)^{-1} + I_N(\lambda I_N)^{-1}I_N\right)^{-1}I_N(\lambda I_N)^{-1}\right]\mathbf{t}
\end{aligned}$$

Dual Representations: Derivation

- Now let's use matrix inversion lemma backward:

$$A^{-1} - A^{-1}U(C^{-1}+VA^{-1}U)^{-1}VA^{-1} = (A+UCV)^{-1}$$

we have

$$\begin{aligned}
\mathbf{w} &= \Phi^T\left[(\lambda I_N)^{-1} - (\lambda I_N)^{-1}I_N\left(\left(\Phi\Phi^T\right)^{-1} + I_N(\lambda I_N)^{-1}I_N\right)^{-1}I_N(\lambda I_N)^{-1}\right]\mathbf{t} \\
&= \Phi^T\left(\lambda I_N + \Phi\Phi^T\right)^{-1}\mathbf{t} \\
&\triangleq \Phi^T\mathbf{a}
\end{aligned}$$

of which $\mathbf{a} = \left(\lambda I_N + \Phi\Phi^T\right)^{-1}\mathbf{t} \in \mathbb{R}^N$

- What is this $\mathbf{a}$?
  - It will take the place of $\mathbf{w}$ and be the parameter in dual representation.
  - We will see shortly

Dual Representations: Derivation

- The prediction is

$$y(\mathbf{x}) = w^T\phi(\mathbf{x}) = \mathbf{a}^T\Phi\phi(x) = \left[\left(\lambda I_N + \Phi\Phi^T\right)^{-1}\mathbf{t}\right]^T\Phi\phi(x)$$

$$= \left[\left(\lambda I_N + \begin{bmatrix}\phi(\mathbf{x}_1)^T\phi(\mathbf{x}_1) & \cdots & \phi(\mathbf{x}_1)^T\phi(\mathbf{x}_N)\\ \vdots & \ddots & \vdots \\ \phi(\mathbf{x}_N)^T\phi(\mathbf{x}_1) & \cdots & \phi(\mathbf{x}_N)^T\phi(\mathbf{x}_N)\end{bmatrix}\right)^{-1}\mathbf{t}\right]^T\begin{bmatrix}\phi(\mathbf{x}_1)^T\phi(\mathbf{x})\\ \vdots \\ \phi(\mathbf{x}_N)^T\phi(\mathbf{x})\end{bmatrix}$$

- Inner product appears! **Replace with kernels**!
- Define

$$K = \Phi\Phi^T = \begin{bmatrix}\kappa(\mathbf{x}_1,\mathbf{x}_1) & \cdots & \kappa(\mathbf{x}_1,\mathbf{x}_1)\\ \vdots & \ddots & \vdots \\ \kappa(\mathbf{x}_N,\mathbf{x}_1) & \cdots & \kappa(\mathbf{x}_N,\mathbf{x}_N)\end{bmatrix} \qquad k(\mathbf{x}) = \Phi\phi(\mathbf{x}) = \begin{bmatrix}\kappa(\mathbf{x}_1,\mathbf{x})\\ \vdots \\ \kappa(\mathbf{x}_N,\mathbf{x})\end{bmatrix}$$

  So we have

$$\boxed{\begin{aligned}\mathbf{a} &= (\lambda I_N + K)^{-1}\mathbf{t}\\ y(\mathbf{x}) &= \mathbf{a}^T k(\mathbf{x})\end{aligned}}$$

- Inner products disappear and data enters only through kernels!
  - This is exactly the **dual representation**!

Dual Representations: Primal vs. Dual

- **Primal**:
  - **Parameter:** $\mathbf{w} = (\Phi^T\Phi + \lambda I_M)^{-1}\Phi^T\mathbf{t}$
  - **Prediction:** $y(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x})$
    - invert $\Phi^T\Phi \in \mathbb{R}^{M\times M}$
    - cheaper since usually $N >> M$
    - must explicitly construct features
- **Dual**:
  - **Parameter:** $\mathbf{a} = (K + \lambda I_N)^{-1}\mathbf{t}$
  - **Prediction:** $y(\mathbf{x}) = \mathbf{a}^T k(\mathbf{x})$
    - invert Gram matrix $K = \Phi\Phi^T \in \mathbb{R}^{N\times N}$
    - use kernel trick to avoid feature construction
    - we could easily replace one kernel with another to represent similarity over vectors, images, sequences, graphs, text, etc..

Kernel Regression

> For more details, see **[PRML]** Chapter 6.3

Kernel Regression

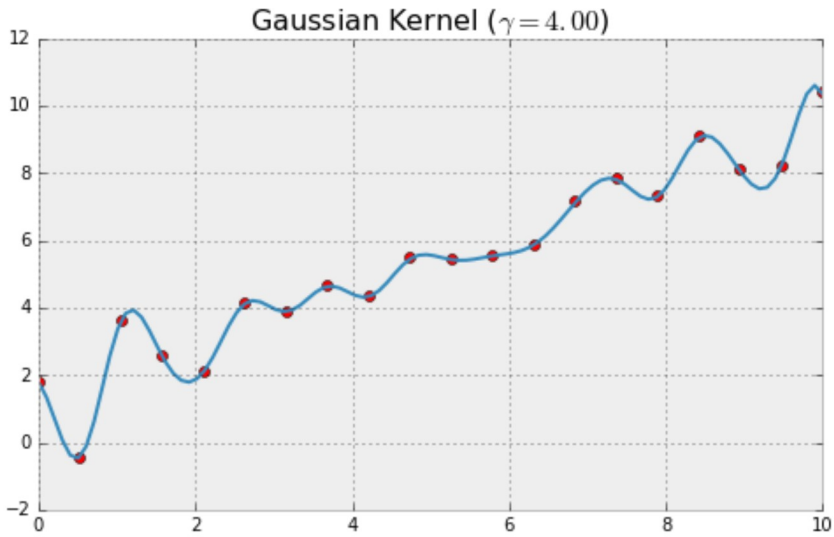
- From training data $\{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N)\}$, **Kernel Regression** outputs:

$$y(\mathbf{x}) = \frac{1}{\sum_{n=1}^N \kappa(\mathbf{x}_n,\mathbf{x})}\sum_{n=1}^N \kappa(\mathbf{x}_n,\mathbf{x})t_n$$

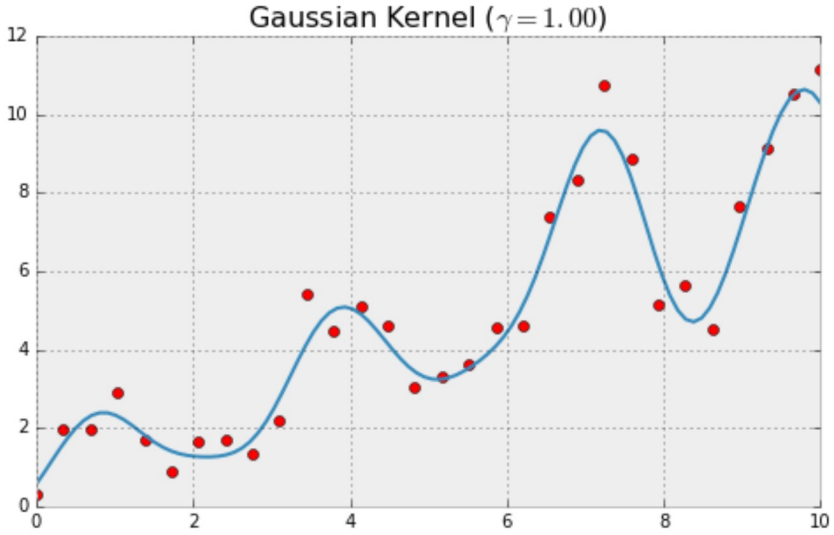
- Approximates $y(\mathbf{x})$ with a weighted sum of nearby points
- Any distance kernel can be used
- A common practice is to use **Gaussian Kernel**:

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

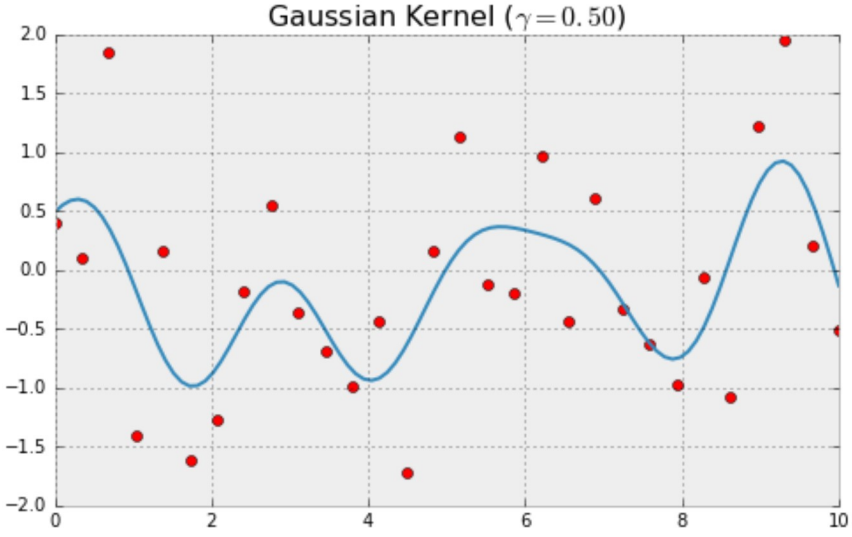
Kernel Regression: Example—Linear + Noise



Kernel Regression: Example—Sine + Linear + Noise



Kernel Regression: Example—Completely Random



Kernel Regression: Classification

- It is very easy to adapt kernel regression to **classification**!
- For data $\mathcal{D} = \{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N)\}$ and kernel $\kappa(\mathbf{x}, \mathbf{x}')$
  - **Regression**: if $t \in \mathbb{R}$, return weighted average:

$$y(\mathbf{x}) = \frac{1}{\sum_{n=1}^{N} \kappa(\mathbf{x}, \mathbf{x}_n)} \sum_{n=1}^{N} \kappa(\mathbf{x}, \mathbf{x}_n) t_n$$

  - **Classification** if $t \in \pm 1$, return weighted majority:

$$h(\mathbf{x}) = \text{sign}\left(\sum_{n=1}^{N} \kappa(\mathbf{x}, \mathbf{x}_n) t_n \right)$$

Comparison to Locally-Weighted Linear Regression

- Review **Locally-weighted Linear Regression**
  - Fit $\mathbf{w}$ to minimize $\sum_{n=1}^{N} r_n (t_n - \mathbf{w}^T \phi(\mathbf{x}_n))^2$ for weights r_n
  - Output $\mathbf{w}^T \phi(\mathbf{x})$
- Standard choice for weight uses Gaussian Kernel,

$$r_n = \exp\left\{ -\frac{||\mathbf{x}_n - \mathbf{x}||^2}{2\sigma^2} \right\}$$

Comparison to Locally-Weighted Linear Regression

- **Similarities:** 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.
- **Differences:**
  - *LWLR:* Weighted regression; slow, but more accurate
  - *KR:* Weighted mean; faster, but less accurate