12. Kernel Methods
STA3142 Statistical Machine Learning

Kibok Lee

Assistant Professor of

Applied Statistics / Statistics and Data Science

Apr 9, 2024

Assignment 2

- Due **Friday 4/12, 11:59pm**
- Topics
  - (Math/Programming) Logistic Regression
  - (Math/Programming) Softmax Regression
  - (Math) Gaussian Discriminant Analysis
  - (Programming) Naïve Bayes for Spam Classification
- Please read the instruction carefully!
  - Submit one pdf and one zip file separately
  - Write your code only in the designated spaces
  - Do not import additional libraries
  - …
- If you feel difficult, consider to take **option 2**.

Midterm

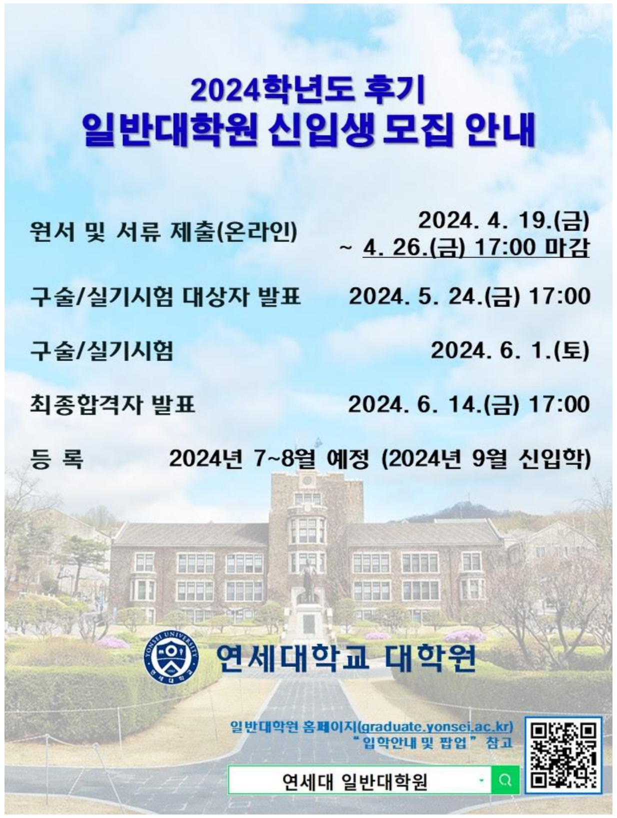
- **Tuesday 4/23, 1:10pm — 2:50pm KST**
  - Please come here by 1:00pm!
  - In-person exam
- Closed book with **an A4-size cheat sheet**
  - You can print/write anything on **both side**.
- Coverage: Lec 6—13
  - True / False, multiple choice, math
- Short practice midterm will be out.
  - To be familiar with the type of midterm questions
  - # questions is about a half of the actual exam
  - **No solution will be provided**

Midterm Coverage

- 4,5: Linear Algebra & Probability Review
  - Not main topics, but you should be familiar with them.
  - Some contents (that we feel difficult) can be given FYI.
- 6,7. Linear Regression (and Other Topics)
- 8. Logistic/Softmax Regression
- 9. Generative Classifiers
- 10. Other Classifiers
- 11. Regularization and Validation
- 12. Kernel Methods
- 13. Support Vector Machines

Announcement

- 24 Fall graduate school application
  - Submission: 4/19 ~ 4/26, 5 PM
  - 1st notification: 5/24, 5 PM
  - Exam: 6/1
  - Final notification: 6/14, 5 PM
  - Registration: July ~ Aug



2024학년도 후기
일반대학원 신입생 모집 안내

원서 및 서류 제출(온라인)	2024. 4. 19.(금) ~ 4. 26.(금) 17:00 마감
구술/실기시험 대상자 발표	2024. 5. 24.(금) 17:00
구술/실기시험	2024. 6. 1.(토)
최종합격자 발표	2024. 6. 14.(금) 17:00
등 록	2024년 7~8월 예정 (2024년 9월 신입학)

연세대학교 대학원

일반대학원 홈페이지(graduate.yonsei.ac.kr)
"입학안내 및 팝업" 참고

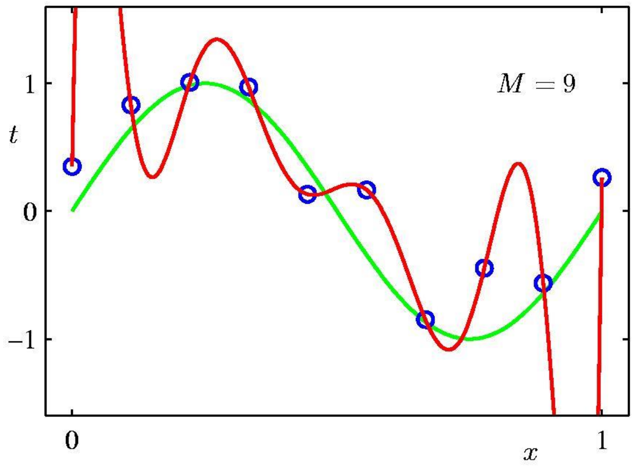
연세대 일반대학원

- More Information/Submission:
  - Kor: https://graduate.yonsei.ac.kr/graduate/index.do
  - Eng: https://graduate.yonsei.ac.kr/graduate_en/index.do

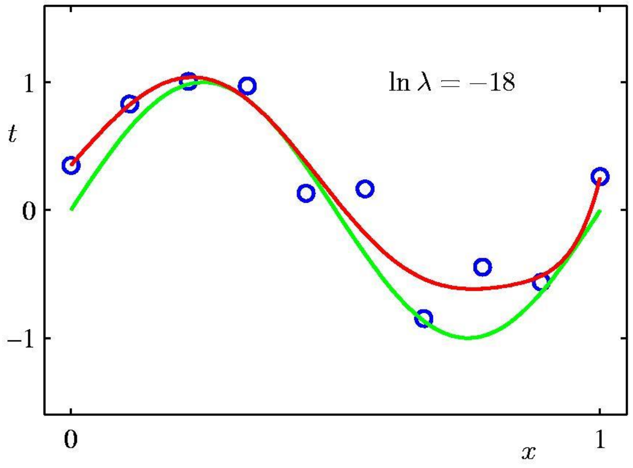
Recap: Regularization

- Regularization controls the tradeoff between "fitting error" and "complexity."
  - Small regularization results in complex models (with risk of overfitting)
  - Large regularization results in simple models (with risk of underfitting)

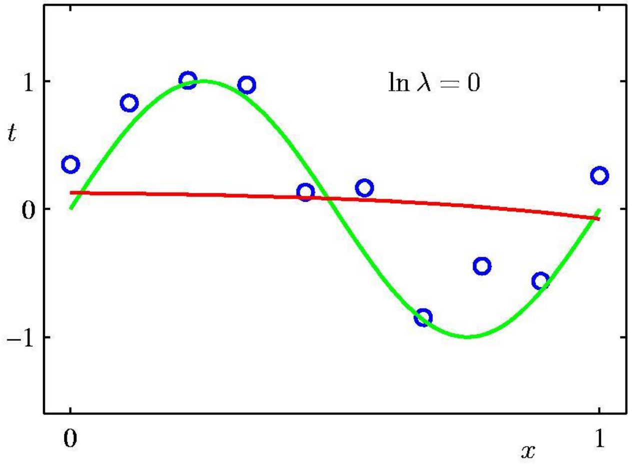
Overfitting



Sweet spot

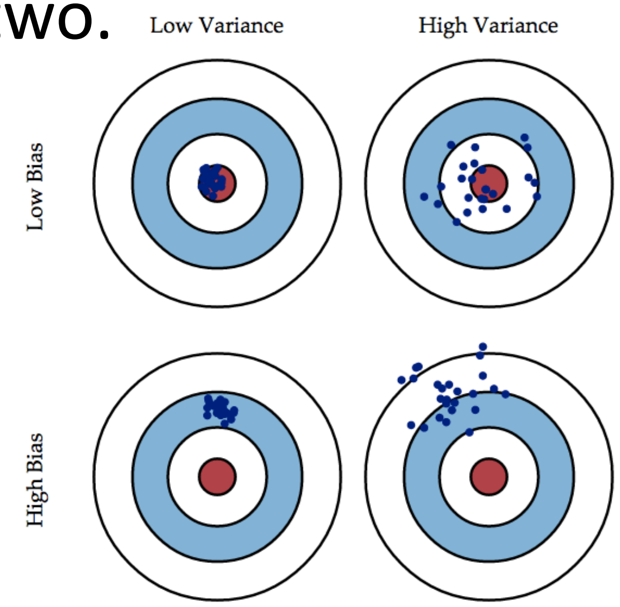


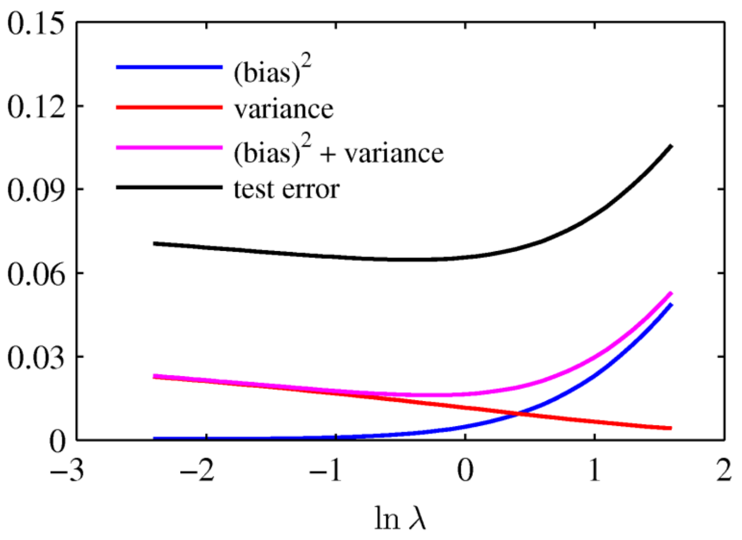
Underfitting



Recap: The Bias-Variance Tradeoff

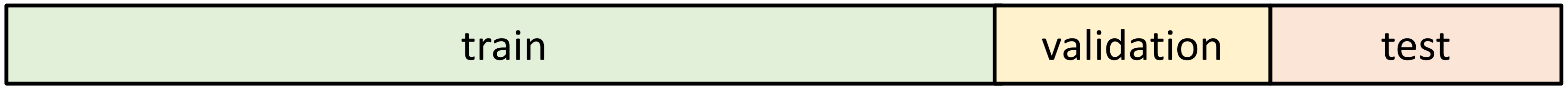
- An over-regularized model (large λ) will have a high bias and low variance.
- An under-regularized model (small λ) will have a high variance and low bias.
- It is important to find a good balance between the two.





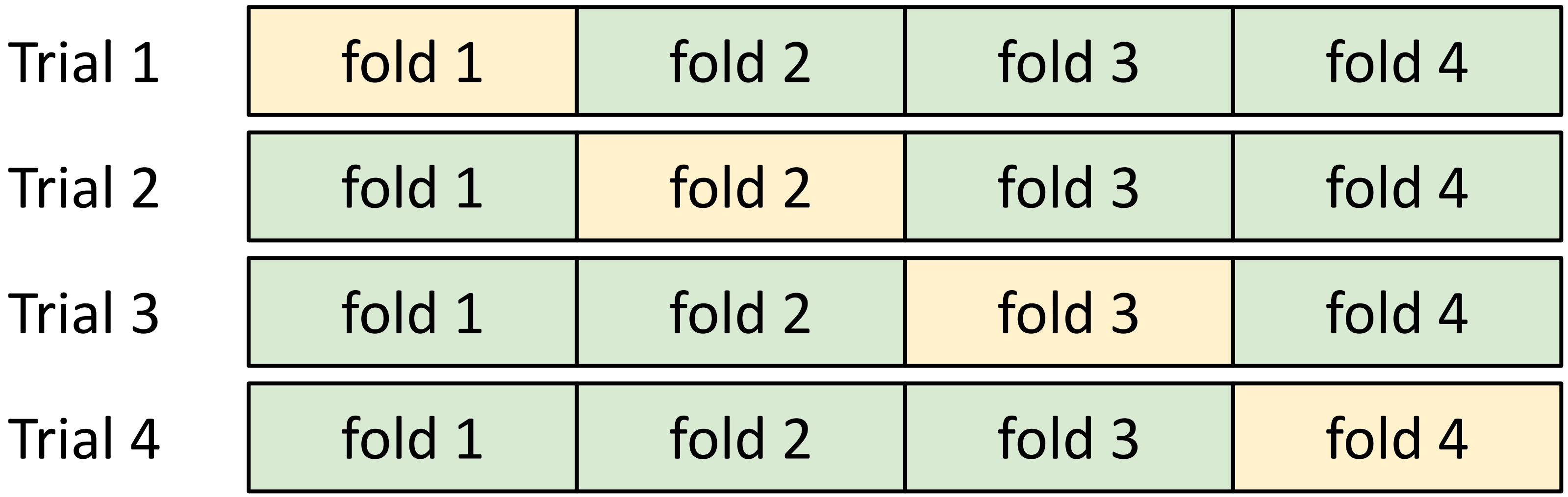
Recap: Validation

- If model selection and true error estimates are to be computed simultaneously, data needs to be divided into three disjoint sets.
- **Training set** to fit the parameters
  - Given a fixed hyperparameters
- **Validation set** to tune/choose the model and hyperparameters
- **Test set** to evaluate the final model performance
  - You must **NOT** tune the model on test set.
  - Test set is **NOT** for model selection.



Recap: K-Fold Cross-Validation

- Split dataset into K-folds
  - Take one fold (yellow) as validation and the rest of K-1 folds (green) for training.



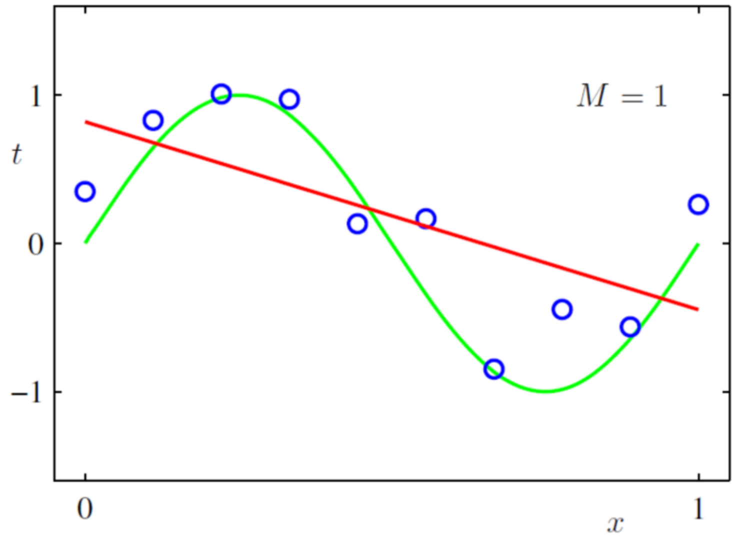
- The final validation error is estimated as the average error rate.

Outline

- Feature Mappings
- Kernel Trick
- Dual Representations
  - Example: Kernel Ridge Regression
- Constructing Kernels
- (Nadaraya-Watson) Kernel Regression

Linear Regression

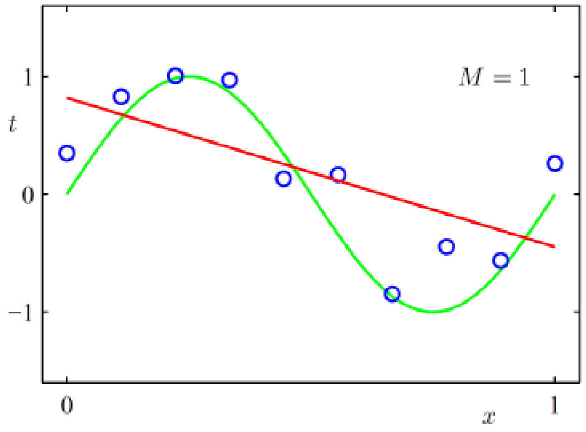
- Linear function $h(\mathbf{x}; \mathbf{w}) = \mathbf{w}^T\mathbf{x}$ can only produce straight lines through origin.
- Affine function $h(\mathbf{x}; \mathbf{w}) = \mathbf{w}^T\mathbf{x} + b$ can only produce straight lines.
  - e.g., 1D linear regression
  - Not very flexible/powerful



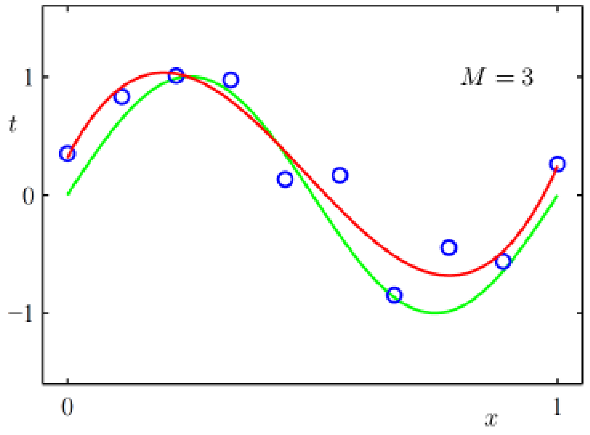
- Can we make it more flexible?

Feature Mappings

- Solution: Add features and hope some of them are useful; e.g., polynomial basis functions
  - $\phi(x) = (1, x)$



  - $\phi(x) = (1, x, x^2, x^3)$



Linear Regression with Features

- Linear regression model

$$h(\mathbf{x};\mathbf{w}) = \mathbf{w}^T\phi(\mathbf{x}) = \sum_{j=0}^{M-1} w_j\phi_j(\mathbf{x})$$

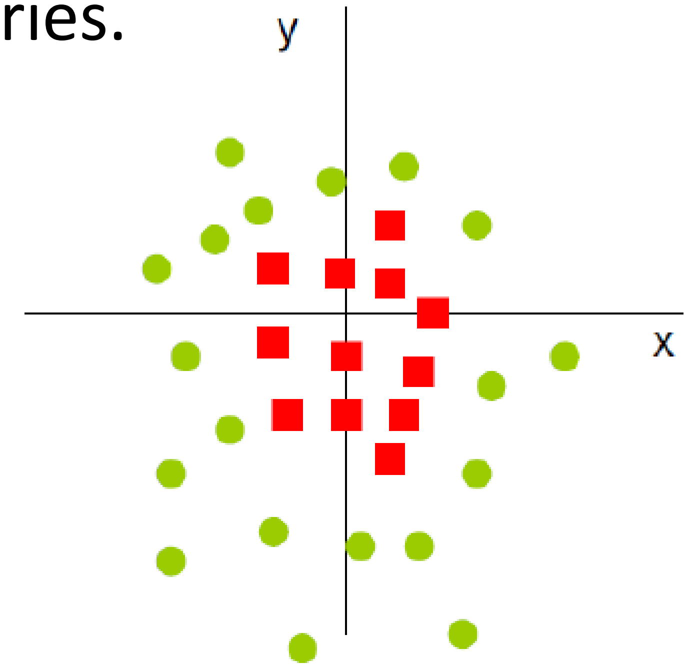
- Least squares with L2 regularization

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

- Gradient: $\nabla_{\mathbf{w}}J(\mathbf{w}) = \Phi^T(\Phi\mathbf{w} - \mathbf{y}) + \lambda\mathbf{w}$
- Closed form solution: $\mathbf{w} = (\Phi^T\Phi + \lambda I)^{-1}\Phi^T\mathbf{y}$

Linear Classifiers

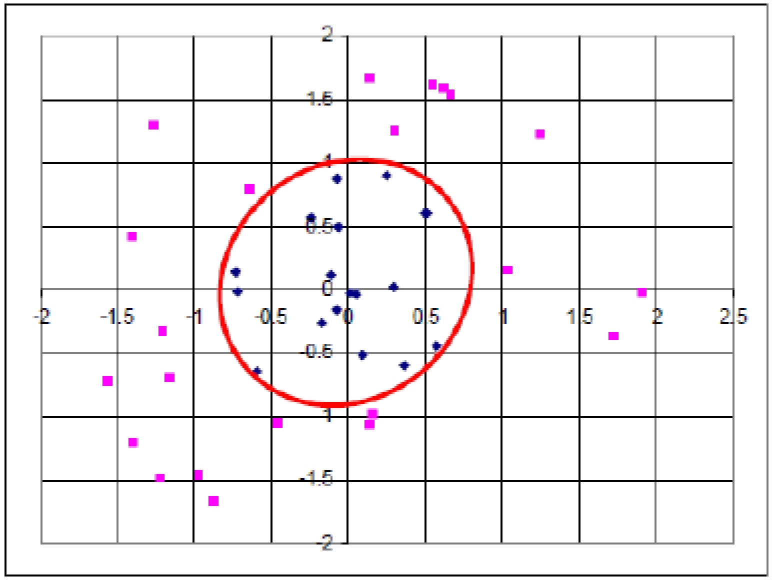
- Linear function $h(\mathbf{x};\mathbf{w}) = \mathbf{w}^T\mathbf{x}$ can only produce linear decision boundaries through origin.
- Affine function $h(\mathbf{x};\mathbf{w}) = \mathbf{w}^T\mathbf{x} + b$ can only produce linear decision boundaries.
  - e.g., 2D linear classification
  - Not very flexible/powerful



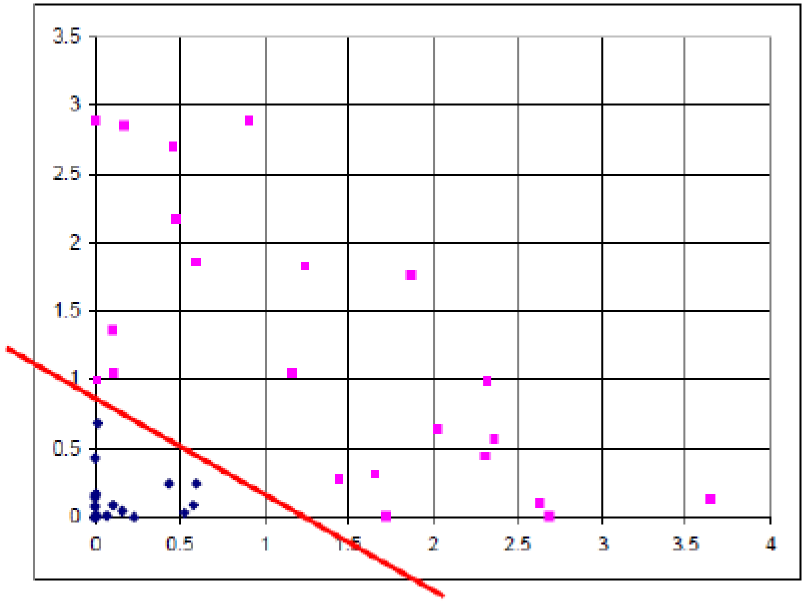
- Can we make it more flexible?

Linear Classifiers with Features

- $\phi(x_1, x_2) = (x_1^2, x_2^2)$



Not linearly separable



Linearly separable

- (Nonlinear) features can make the problem solvable with linear methods.

Feature Selection

- We have been mapping data $\mathbf{x}$ through a fixed (nonlinear) mapping to get a feature vector $\phi(\mathbf{x})$.
  - The feature vector extracts important properties from $\mathbf{x}$.
  - e.g., Polynomial combinations up to some order
  - It makes many regression/classification problems easier.

- In general, what features should we use?

Feature Selection

- How about to use **all features** we can think of?
- Model complexity might not be an issue, as proper regularization can handle it.
- However, it is **not scalable**; assuming large $N \approx M$,
  - N: Number of training data
  - M: Number of features
- Closed-form solution requires $O(N^3)$
- Gradient descent requires $O(N^2)$

Feature Selection

- We have been mapping data $\mathbf{x}$ through a fixed (nonlinear) mapping to get a feature vector $\phi(\mathbf{x})$.
- In other words, with feature mappings, all data have been mapped to a higher dimensional space.
- Alternatively, we can think of it like:
  Data still lives in the original space, but the definition of **distance** or **inner product** has been changed.

Kernel Trick

Kernel Trick

- As we have done, we will embed data $\mathbf{x}$ in a high dimensional space $\phi(\mathbf{x})$, and use simple (linear) models in this space.

- Use algorithms that do not need the coordinates of embeddings $\phi(\mathbf{x})$, but pairwise **inner products**:
$$\phi(\mathbf{x})^T\phi(\mathbf{x}')$$

- Replace these inner products with a **kernel**:
$$k(\mathbf{x},\mathbf{x}') = \phi(\mathbf{x})^T\phi(\mathbf{x}')$$

Kernel Trick

- A **kernel** function $k(\mathbf{x}, \mathbf{x}')$ represents the similarity between $\mathbf{x}$ and $\mathbf{x}'$.

- A popular way to express the similarity between feature vectors is the **inner product** of them:
$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$$

- A **kernel** function $k(\mathbf{x}, \mathbf{x}')$ is defined to be an inner product of feature vectors, but **we do not actually compute them.**

Example: Kernels for 2D Data

- Inner product between (x_1, x_2) and (z_1, z_2):
$$k(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T \mathbf{z} = x_1 z_1 + x_2 z_2$$

- Its square is also a kernel:
$$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T \mathbf{z})^2 = (x_1 z_1 + x_2 z_2)^2$$
$$= x_1^2 z_1^2 + 2 x_1 z_1 x_2 z_2 + x_2^2 z_2^2$$
- This is the same as inner product between $(x_1^2, \sqrt{2} x_1 x_2, x_2^2)$ and $(z_1^2, \sqrt{2} z_1 z_2, z_2^2)$
  - Or, between $(x_1^2, x_1 x_2, x_1 x_2, x_2^2)$ and $(z_1^2, z_1 z_2, z_1 z_2, z_2^2)$
  - **Note: Solution is not unique.**

Example: Kernels for 2D Data

- Consider higher-order polynomial of degree p:

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

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

$\xleftarrow{}$ Multinomial expansion

- Feature mapping:

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

  - All monomials of degree p

Example: Kernels for 2D Data

- Inhomogeneous polynomial up to degree p:

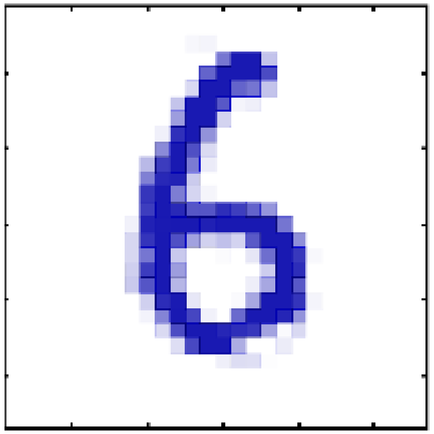
$$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T \mathbf{z} + c)^p = \left(c + \sum_{i=1}^{M} x_i z_i \right)^p, c > 0$$

- Feature mapping:

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

Example: Handwritten Digits

- An image consists of $28 \times 28 = 784$ pixels
$$\mathbf{x} \in [0,1]^{784}$$



- Take the pixel values and compute
$$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^T \mathbf{z} + 1)^p$$

- For $p = 4$, computing the inner product in the space of all monomials **without kernel trick** requires **16G** dimensional space.

Kernel Trick

- With kernels, inner products in a high dimensional space can be computed with the computational complexity of a low dimensional space.
- Many algorithms can be expressed completely in terms of kernels $k(\mathbf{x}, \mathbf{x}')$ without $\phi(\mathbf{x})$.
- We can replace a kernel with another to get a new algorithm that works over a different domain.

Kernel Trick

- To use the kernel trick, we must formulate (training and test) algorithms purely in terms of inner products between data points.
  - All operations w.r.t. $\mathbf{x}$ should look like $k(\mathbf{x}, \mathbf{x}')$
- We cannot access the coordinates in the high-dimensional feature space, i.e., no explicit $\phi(\mathbf{x})$.
- This seems a huge limitation, but many operations/algorithms work under this condition.

Example: Distance

- Distance between two samples can be expressed in inner products:

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

- e.g., K-nearest neighbors can be done in an arbitrary high dimensional space with the kernel trick.

Example: Mean

- Mean of data points:

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

- We cannot determine the mean of data in the mapped feature space through kernel operations.

Example: Distance to Mean

- Mean of data points:

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

- Distance to mean:

$$\begin{aligned}
&\left\|\phi(\mathbf{x}) - \bar{\phi}\right\|^2 \\
&= \phi(\mathbf{x})^T\phi(\mathbf{x}) + \bar{\phi}^T\bar{\phi} - 2\phi(\mathbf{x})^T\bar{\phi} \\
&= k(\mathbf{x},\mathbf{x}) + \frac{1}{N^2}\sum_{i,j=1}^{N} k\left(\mathbf{x}^{(i)},\mathbf{x}^{(j)}\right) - \frac{2}{N}\sum_{i=1}^{N} k\left(\mathbf{x},\mathbf{x}^{(i)}\right)
\end{aligned}$$

Dual Representations

Dual Representations and Kernel Trick

- The dual representation and its solution are entirely written in terms of kernels.
  - The elements of the Gram matrix $K = \Phi\Phi^T$

$$K_{ij} = \phi\left(\mathbf{x}^{(i)}\right)^T \phi\left(\mathbf{x}^{(j)}\right) = k\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right)$$

- These represent the pairwise similarities among all the observed feature vectors.
  - We can compute kernels more efficiently than the feature vectors.

Example: Kernel Ridge Regression

- Recall regression problems with error function

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

- $J(\mathbf{w})$ is minimized at

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

- Recall the $N \times M$ data matrix that is central to this solution.

Recap: The Data Matrix

- The data matrix is an $N \times M$ matrix, applying
  - the M basis functions (columns)
  - to N data points (rows)

$$\Phi = \begin{pmatrix} \phi_0(\mathbf{x}^{(1)}) & \phi_1(\mathbf{x}^{(1)}) & \dots & \phi_{M-1}(\mathbf{x}^{(1)}) \\ \phi_0(\mathbf{x}^{(2)}) & \phi_1(\mathbf{x}^{(2)}) & \dots & \phi_{M-1}(\mathbf{x}^{(2)}) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\mathbf{x}^{(N)}) & \phi_1(\mathbf{x}^{(N)}) & \dots & \phi_{M-1}(\mathbf{x}^{(N)}) \end{pmatrix}$$

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

The Gram Matrix

- For closed-form solution, we use the $M \times M$ (scaled covariance) matrix: $\Phi^T\Phi$

- For dual representation, we use the $N \times N$ Gram matrix: $K = \Phi\Phi^T$
  - where $K_{ij} = \phi\left(\mathbf{x}^{(i)}\right)^T \phi\left(\mathbf{x}^{(j)}\right) = k\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right)$, each element corresponds to the pairwise similarities of two training data.

- Note that kernel methods use only K, not Φ.

Example: Kernel Ridge Regression

- Objective function:

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

- Take derivative: $\nabla_{\mathbf{w}}J(\mathbf{w}) = \Phi^T(\Phi\mathbf{w} - \mathbf{y}) + \lambda\mathbf{w} = 0$
- Then,

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

  - where $\mathbf{a} = [a_1, \dots, a_N]^T$; $a_n = -\frac{1}{\lambda}\left(\mathbf{w}^T\phi\left(\mathbf{x}^{(n)}\right) - y^{(n)}\right)$ be the new parameters.
- i.e., transform $J(\mathbf{w})$ to $J(\mathbf{a})$ with $\mathbf{w} = \Phi^T\mathbf{a}$

Example: Kernel Ridge Regression

- Objective function:

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

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

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

$$= \frac{1}{2}\mathbf{a}^T KK\mathbf{a} - \mathbf{a}^T K\mathbf{y} + \frac{1}{2}\mathbf{y}^T\mathbf{y} + \frac{\lambda}{2}\mathbf{a}^T K\mathbf{a}$$

- Solution: $\mathbf{a} = (K + \lambda I_N)^{-1}\mathbf{y}$
- At test time:

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

  - where $\mathbf{k}(\mathbf{x}) = \left[k\left(\mathbf{x}^{(1)}, \mathbf{x}\right), \dots, k\left(\mathbf{x}^{(N)}, \mathbf{x}\right)\right]^T$

Primal vs. Dual Representations

- Primal: $\mathbf{w} = (\Phi^T\Phi + \lambda I_M)^{-1}\Phi^T\mathbf{y}$
  - Need to invert an $M \times M$ matrix, where M is the feature dimension
  - Efficient when $N > M$
  - Need to compute features explicitly

- Dual: $\mathbf{a} = (K + \lambda I_N)^{-1}\mathbf{y}$
  - Need to invert an $N \times N$ matrix, where N is the number of training data
  - Efficient when $N < M$
  - Kernel trick is applicable; kernels can be defined over vectors, images, sequences, graphs, text, etc.

Constructing Kernels

Constructing Kernels: Method 1

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

- Define a feature mapping $\phi(\mathbf{x})$ and then define the inner product of features as kernel (or vice versa)
  - Formally, $k: \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a valid kernel if and only if there exists $\phi: \mathcal{X} \to \mathcal{H}$ such that
    $$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}'), \forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}$$
  - where $\mathcal{H}$ is a Hilbert space (= Euclidean space with potentially infinite-dimensional)

Constructing Kernels: Method 1

- Define a kernel function directly, such as
$$k(\mathbf{x}, \mathbf{z}) = \mathbf{x}^T \mathbf{z} = x_1 z_1 + x_2 z_2$$

- In 2D, we can explicitly identify the feature map
$$\phi(\mathbf{x}) = \left(x_1^2, \sqrt{2} x_1 x_2, x_2^2\right)$$
  - 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: Method 2

- A simpler way to test if a given function is kernel without constructing $\phi(\mathbf{x})$

1. Show that the Gram matrix K is positive semidefinite (PSD) for all possible choices of the dataset.

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

2. Use **Mercer's theorem** to prove that the kernel function $k(\mathbf{x}, \mathbf{x}')$ is a valid kernel
   - Mercer's theorem is **the necessary and sufficient condition**: K is PSD iff $k(\mathbf{x}, \mathbf{x}')$ is a valid kernel

Constructing Kernels: Method 3

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

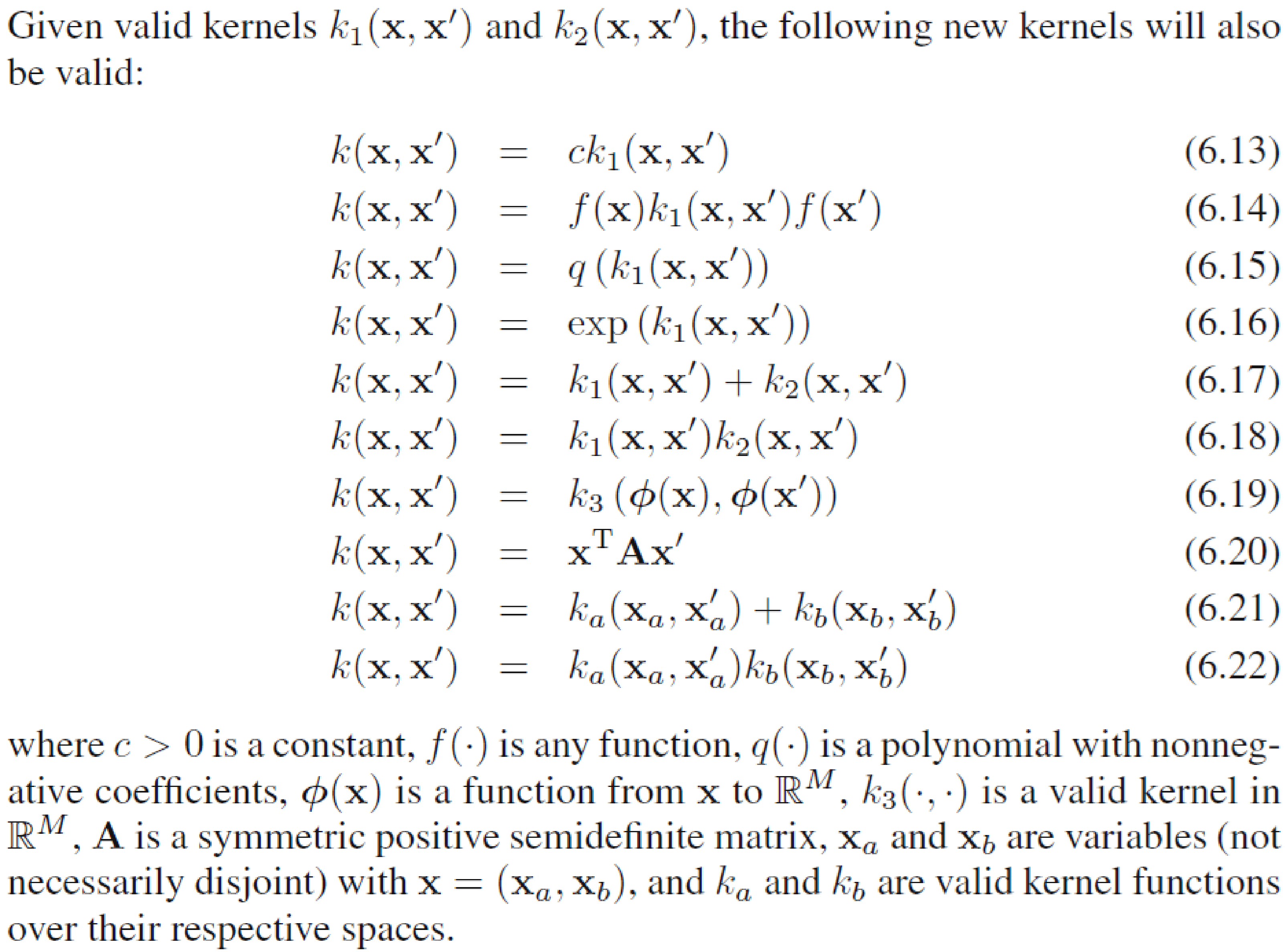
- For example, with a known kernel k_1,

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

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

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

Constructing Kernels: Method 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:

$$k(\mathbf{x}, \mathbf{x}') = ck_1(\mathbf{x}, \mathbf{x}') \tag{6.13}$$

$$k(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})k_1(\mathbf{x}, \mathbf{x}')f(\mathbf{x}') \tag{6.14}$$

$$k(\mathbf{x}, \mathbf{x}') = q\left(k_1(\mathbf{x}, \mathbf{x}')\right) \tag{6.15}$$

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(k_1(\mathbf{x}, \mathbf{x}')\right) \tag{6.16}$$

$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') \tag{6.17}$$

$$k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}') \tag{6.18}$$

$$k(\mathbf{x}, \mathbf{x}') = k_3\left(\boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}')\right) \tag{6.19}$$

$$k(\mathbf{x}, \mathbf{x}') = \mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x}' \tag{6.20}$$

$$k(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x}_a, \mathbf{x}_a') + k_b(\mathbf{x}_b, \mathbf{x}_b') \tag{6.21}$$

$$k(\mathbf{x}, \mathbf{x}') = k_a(\mathbf{x}_a, \mathbf{x}_a')k_b(\mathbf{x}_b, \mathbf{x}_b') \tag{6.22}$$

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.

How to Prove If a Kernel Is Valid?

1. Prove that there exists a ϕ such that
$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}'), \forall \mathbf{x}, \mathbf{x}'$$
2. Prove that the Gram matrix K is PSD and use Mercer’s Theorem
   - Note: $\mathrm{PSD} + \mathrm{PSD} = \mathrm{PSD}$ and $c \times \mathrm{PSD} = \mathrm{PSD}$ for $c \geq 0$
   - Also useful to prove if a kernel is invalid; provide a counterexample showing that the Gram matrix K is not PSD
3. Use the axioms provided in previous slides
   - But **not for assignments & exams**; you need to prove them before using them.

Commonly Used Kernels

- Simple polynomial kernel (degree of 2)

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

- Generalized polynomial kernel (degree of M)

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

- Gaussian kernel

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

- Cf. Radial basis function (RBF) kernel is essentially the same, but has different parametrization.

$$k(\mathbf{x}, \mathbf{z}) = \exp(-\gamma ||\mathbf{x} - \mathbf{z}||^2)$$

Gaussian Kernel

- Not actually Gaussian pdf

- Translation invariant; depends only on distance between points, such that it can be expressed as
$$k(\mathbf{x}, \mathbf{z}) = k(\mathbf{x} - \mathbf{z})$$

- Corresponds to a mapping to an infinite-dimensional space!

(Nadaraya-Watson) Kernel Regression

Kernel Regression

- Intuition: If a test data is close to a training data, then they would have a similar target.
  - Take a kernel as the similarity measure
  - Take the weighted average of targets of training data, where the weight is the kernel value
    - No training is required, similar to K-nearest neighbors
- Kernel regression output

$$h(\mathbf{x}) = \frac{\sum_n k\left(\mathbf{x}, \mathbf{x}^{(n)}\right) y^{(n)}}{\sum_n k\left(\mathbf{x}, \mathbf{x}^{(n)}\right)}$$

- Note that this is different from kernel ridge regression.

Kernel Regression vs. Classification

- Similar to K-nearest neighbors, we can reformulate it into classification.

- Kernel regression

$$h(\mathbf{x}) = \frac{\sum_n k\left(\mathbf{x}, \mathbf{x}^{(n)}\right) y^{(n)}}{\sum_n k\left(\mathbf{x}, \mathbf{x}^{(n)}\right)}$$

- Kernel classification (for $y \in \{-1, +1\}$)

$$h(\mathbf{x}) = \operatorname{sign}\left(\sum_n k\left(\mathbf{x}, \mathbf{x}^{(n)}\right) y^{(n)}\right)$$

vs. Locally-Weighted Linear Regression

- Kernel regression
  - No training
  - Output: $h(\mathbf{x}) = \frac{\sum_n k(\mathbf{x},\mathbf{x}^{(n)}) y^{(n)}}{\sum_n k(\mathbf{x},\mathbf{x}^{(n)})}$
- Locally-weighted linear regression
  - Find $\mathbf{w}$ to minimize $J(\mathbf{w}) = \sum_n r^{(n)} \left(\mathbf{w}^T \mathbf{x}^{(n)} - y^{(n)}\right)^2$
  - Output: $h(\mathbf{x}) = \mathbf{w}^T \mathbf{x}^{(n)}$
- Common choice for both $k(\mathbf{x}, \mathbf{x}^{(n)})$ and $r^{(n)}$ is Gaussian kernel:

$$\exp\left(-\frac{\left\|\mathbf{x} - \mathbf{x}^{(n)}\right\|^2}{2\sigma^2}\right)$$

vs. Locally-Weighted Linear Regression

- Similar: Instance-based learning
  - Only observations (training data) close to the query point are considered (highly weighted) for regression.
  - Kernel determines how much weights to training data by computing similarity to the query (test data).
  - Free to choose any kernel
  - Both can suffer when the input dimension is high.
- Dissimilar:
  - Kernel regression does not perform training.
  - In general, kernel regression is faster but less accurate.

Next: Support Vector Machines