7. Multi-class Classification and Kernel Methods

Dongwoo Kim

dongwoo.kim@postech.ac.kr

CSED515 - 2023 Spring

Application of Classification

Binary classification

- Image classification: Is the image showing a cat? Yes/No
- Edge detection: Is a given pixel at a boundary? Yes/No
- Object detection: Is there a cat in the image? Yes/No
- ...

Multi-class classification

- Which object is illustrated? Cat/Dog/Tiger/Zebra...

Table of Contents

Recall) Discriminant Function

A description of classifier:

$$\text{assign } x \text{ to } \mathcal{C}_k \text{ if } f_k(x) > f_j(x) \text{ for all } j \neq k$$

- $\{f_k\}_{k=1,\ldots,K}$: discriminant function/score
  - e.g., $f_k(x) = p(\mathcal{C}_k \mid x)$
- i.e., classifier $h(x) = \arg\max_k f_k(x)$
- If $f_k(\cdot)$'s are a set of discriminant functions, then for any monotonically increasing function $g : \mathbb{R} \to \mathbb{R}$ $g_k(\cdot) = g(f_k(\cdot))$'s are also a set of discriminant functions
  - e.g.1, $a\, f_k(x) + b$ for any $a > 0$ and b
  - e.g.2, $\log f_k(\cdot)$
  - ...

Recall) Discriminant Function: Bayes Decision Rule

Bayes decision rule:

$$f_k(x) = p(\mathcal{C}_k \mid x) \text{ or equivalently, } f_k(x) = p(x \mid \mathcal{C}_k)p(\mathcal{C}_k)$$

- Classify $x \in \mathcal{C}_k$ if $p(\mathcal{C}_k \mid x) > p(\mathcal{C}_j \mid x)$ for all $j \neq k$.
- We can use discriminant function $f_k(x) = p(x \mid \mathcal{C}_k)p(\mathcal{C}_k)$ without normalization factor thanks to Bayes rule:

$$p(\mathcal{C}_k \mid x) = \frac{\overbrace{p(x \mid \mathcal{C}_k)}^{\text{Class-conditional density}} \overbrace{p(\mathcal{C}_k)}^{\text{Prior}}}{\underbrace{\sum_{j=1}^{K} p(x \mid \mathcal{C}_j)p(\mathcal{C}_j)}_{\text{Normalization factor}}}$$

In general, a classification task can be understand as a task of learning $p(y = k \mid x)$ (or score function) for each class k

Multi-class Logistic Regression for Classification

- A model of multinomial distribution over K-class $y \in \{1, 2, ..., K\}$:
  - Using K weight vectors, denoted by $\boldsymbol{w}_{(y)}$'s, for every class

$$p(y = k|x) = \frac{\exp(\boldsymbol{w}_{(k)}^\top \phi(x))}{\sum_{j=1}^{K} \exp(\boldsymbol{w}_{(j)}^\top \phi(x))}$$

  - Classifier (Bayes decision rule):

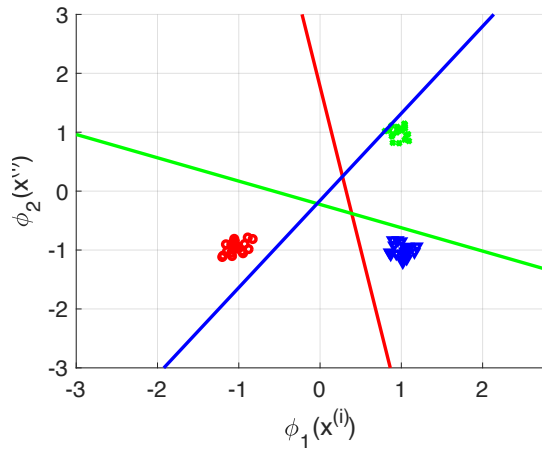
$$\arg\max_k p(y = k|x) = \arg\max_k \boldsymbol{w}_{(k)}^\top \phi(x)$$

- Learning $\boldsymbol{w} = (\boldsymbol{w}_{(y)})_{y=1,...,K}$ is maximizing the likelihood (assuming i.i.d. data points) as before:

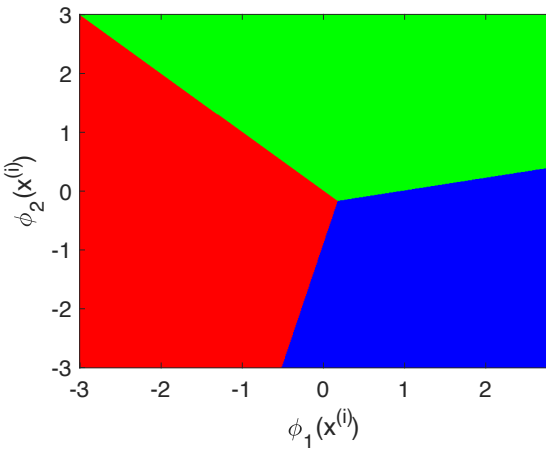
$$\arg\max_{\boldsymbol{w}} \prod_{i=1,...,N} p(y = y_i|x^{(i)}) = \arg\min_{\boldsymbol{w}} -\log p(y = y_i|x^{(i)})$$

An Example of Multi-class Logistic Regression

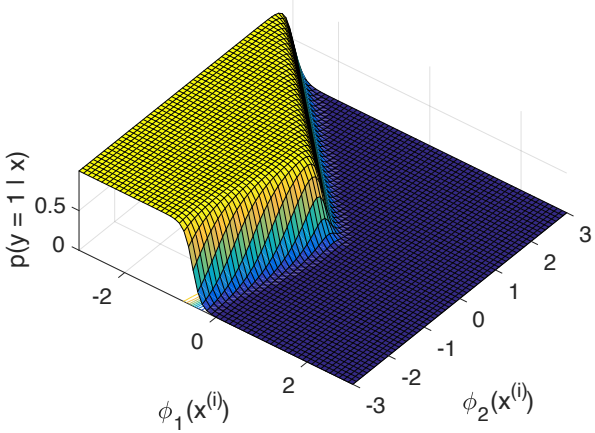
Setup



Decision Boundary



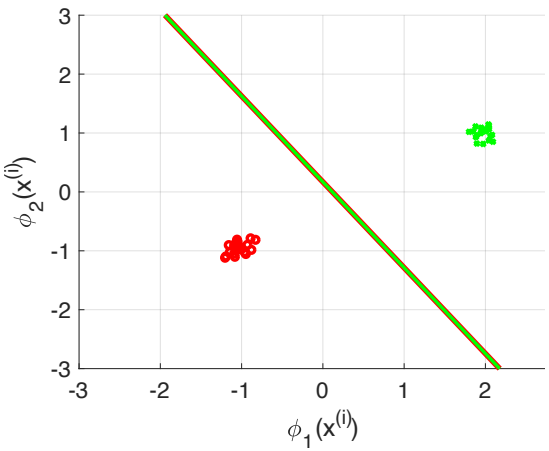
Probability for $k = 1$



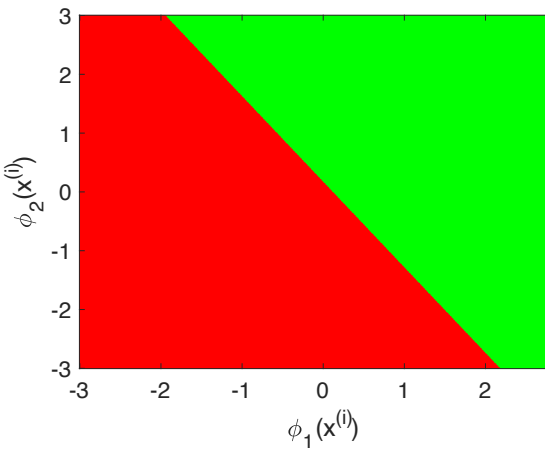
An Example of Multi-class Logistic Regression

What if $K = 2$?

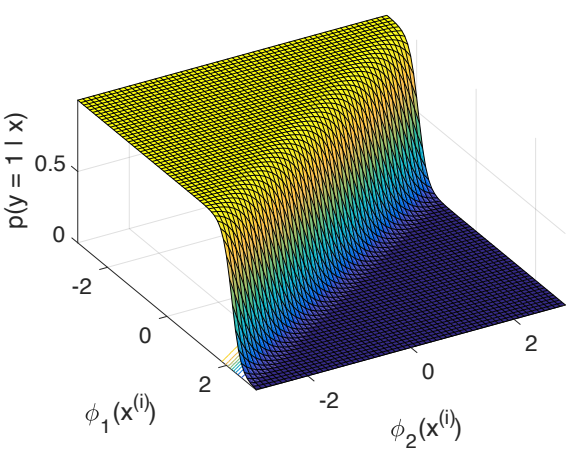
Setup



Decision Boundary



Probability for $k = 1$

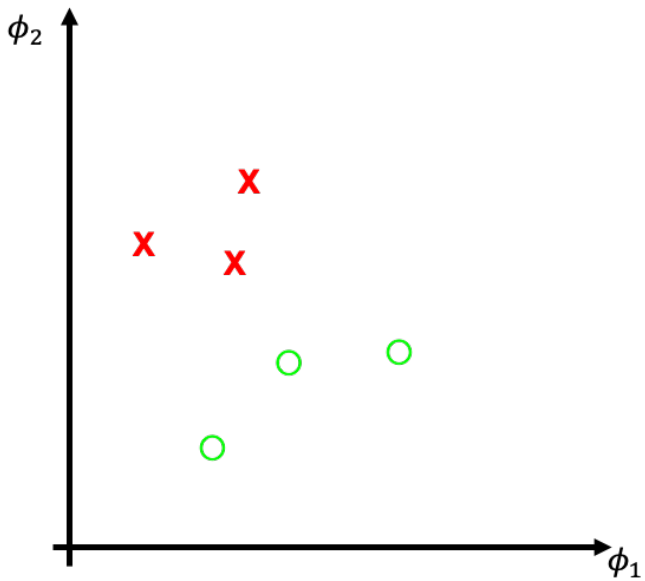


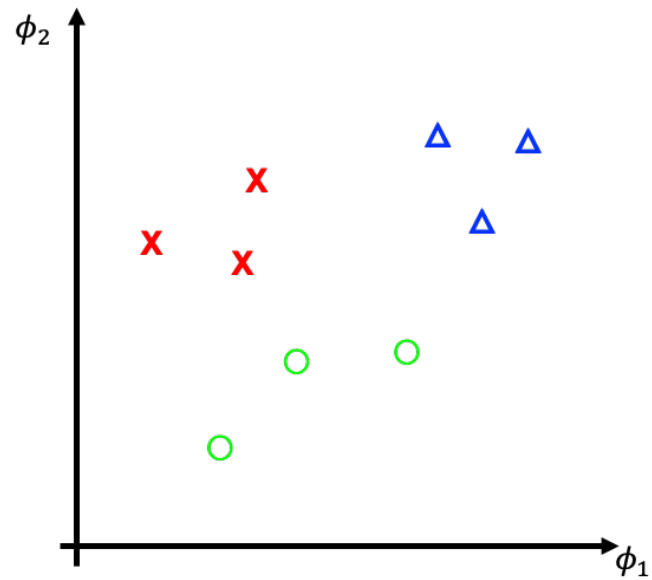
We have $\boldsymbol{w}_{(1)} = -\boldsymbol{w}_{(2)}$

Table of Contents

Binary SVM: Drawing Hyperplane

Combination of binary SVMs = K-class SVM?

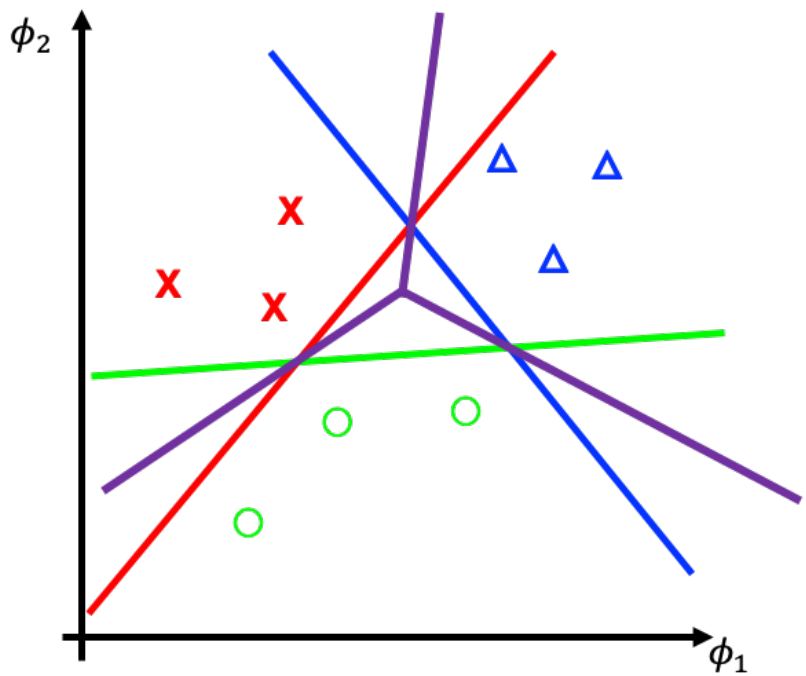




Approach1: One vs. Rest (OvR)

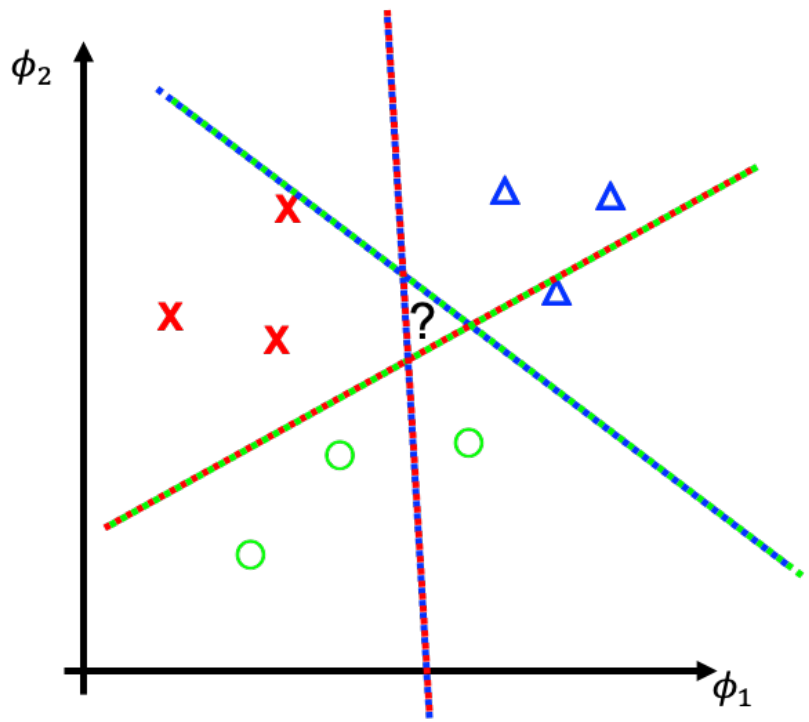
One-vs-Rest, OvR; One-vs-All, OvA; or One-against-All, OAA

- Learn K SVM's, each of which is denoted by $(\boldsymbol{w}_{(k)}, b_{(k)})$ for binary classification between class k (positive) and others (negative)
- For new data $\boldsymbol{x}$, output $\arg\max_{k\in[K]}(\boldsymbol{w}_{(k)}^{\top}\phi(\boldsymbol{x}) + b_{(k)})$



Approach2: One vs. (another) One (OvO)

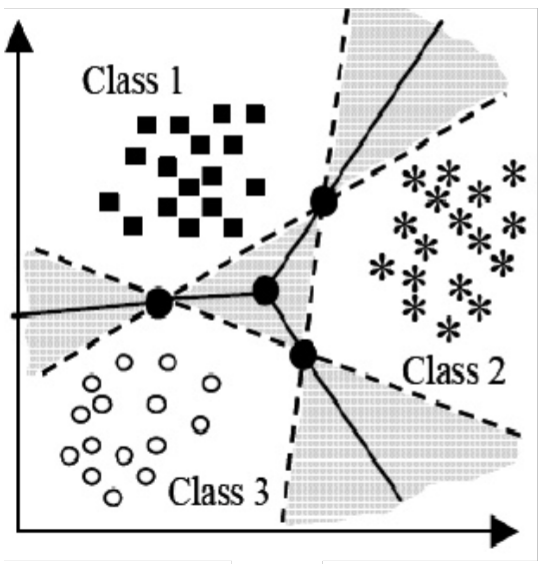
- Learn $K(K-1)/2$ SVM's, each of which is denoted by $(\boldsymbol{w}_{(jk)}, b_{(jk)})$ binary classification between two different classes j and k, distinguishing classes k and j
- For new data $\boldsymbol{x}$, take the majority vote of $K(K-1)/2$ SVM's with some tie-breaking rule



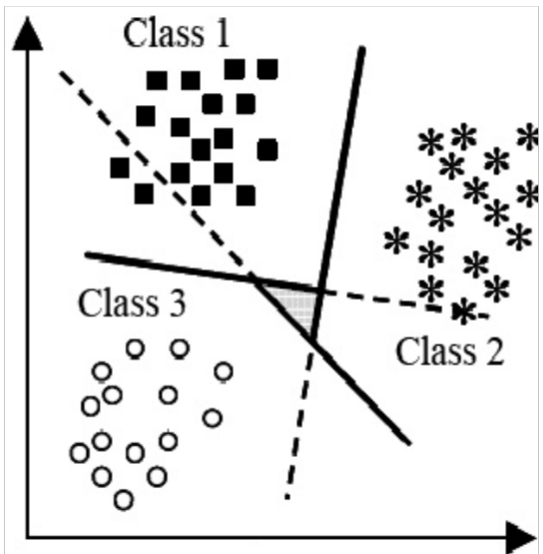
OvR and OvO SVM

- OvR SVM has class imbalance problems as a class is much smaller than the others.
- Each of OvO SVM learns a part of dataset and computationally expensive.

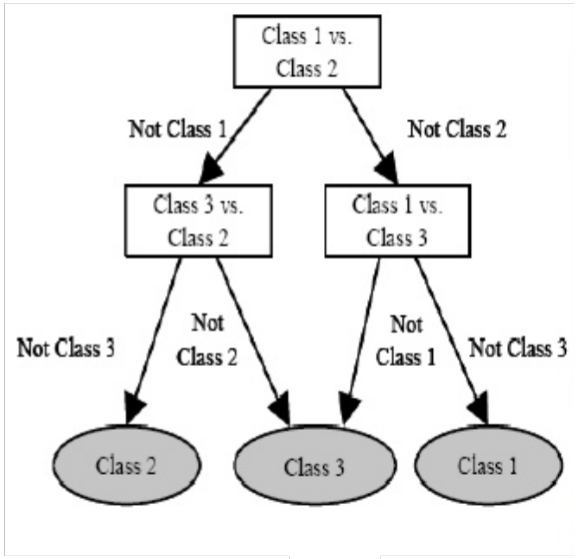
OvR SVM



OvO SVM



DAGSVM



Approach3 by Weston & Watkins 1999 (WW-SVM)

- Solve

$$\min_{\boldsymbol{w},\boldsymbol{b},\boldsymbol{\zeta}\geq 0} \quad \frac{1}{2}\sum_{k=1}^{K}\|\boldsymbol{w}_{(k)}\|_2^2 + C\sum_{i=1}^{N}\sum_{k\neq y_i}\zeta_{(k)}^{(i)}$$

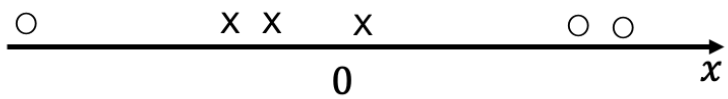
$$\text{s.t.} \quad \boldsymbol{w}_{(y_i)}^\top\phi(\boldsymbol{x}_i) + b_{(y_i)} - (\boldsymbol{w}_{(k)}^\top\phi(\boldsymbol{x}_i) + b_{(k)}) \geq 2 - \zeta_{(k)}^{(i)}$$

- For new data $\boldsymbol{x}$, output $\arg\max_{k\in[K]}(\boldsymbol{w}_{(k)}^\top\phi(\boldsymbol{x}) + b_{(k)})$
- This may resolve the imbalance issue in OvR SVM, and the local-view issue in OvO SVM
- What are the dual problem, and the support vectors in WW SVM? c.f.,

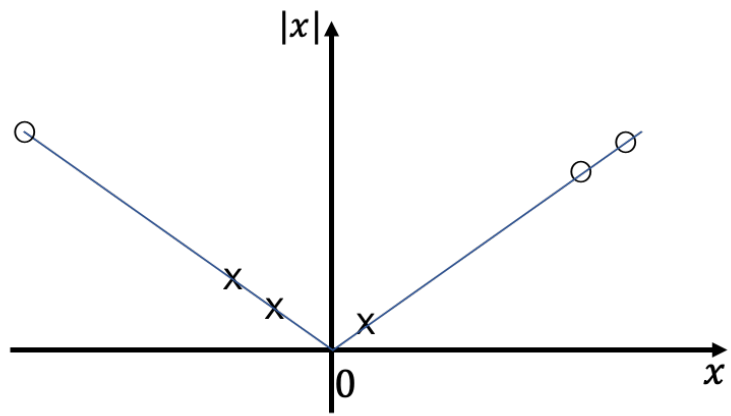
  https://www.csie.ntu.edu.tw/∼cjlin/papers/multisvm.pdf

Are We Done with SVM?

- Still, the representation power of SVM is limited to hyperplanes
  - Option1. allow slack ($\zeta^{(i)}$'s) a.k.a. soft-margin SVM for dataset which are not linearly separable
  - Option2. extract more/other features ϕ or use kernel trick



Not linearly separable



Linearly separable

Table of Contents

Dual Problem of Binary Soft-Margin SVM - L1 Norm

Dual of soft-margin SVM with ℓ_1 norm:

$$\max_{\boldsymbol{\alpha}} \quad -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j \boldsymbol{\phi}^\top(\boldsymbol{x}_i)\boldsymbol{\phi}(\boldsymbol{x}_j) + \sum_{i=1}^{N}\alpha_i$$

$$\text{s.t.} \quad \sum_{i=1}^{N}\alpha_i y_i = 0$$

$$0 \leq \alpha_i \leq C \quad \forall i = 1, ..., N$$

- Note that the problem depends on $\boldsymbol{\phi}^\top(\boldsymbol{x}_i)\boldsymbol{\phi}(\boldsymbol{x}_j)$'s
- Further more, the solution also depends on $\boldsymbol{\phi}^\top(\boldsymbol{x}_i)\boldsymbol{\phi}(\boldsymbol{x}_j)$'s
- Kernel trick: use $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$ instead of $\boldsymbol{\phi}^\top(\boldsymbol{x}_i)\boldsymbol{\phi}(\boldsymbol{x}_j)$

Kernel Trick

- Computing exact feature $\phi(\boldsymbol{x})$ for the entire dataset could be expensive.
  - i.e. $\phi(x) = [1, x_1, x_2, \cdots, x_1^2, x_1x_2, x_1x_3, ..., x_1^3, x_1^2x_2, \cdots]^\top$
  - The dimension of $\boldsymbol{w}$ increases as we expand the feature map.
- Kernel trick: replace the inner product $\phi(\boldsymbol{x})^\top \phi(\boldsymbol{x}')$ by kernel function, $\kappa(\boldsymbol{x}, \boldsymbol{x}')$
- We don't need to compute exact feature map $\phi(\boldsymbol{x})$ given kernel function.
  - The number of Lagrangian multipliers α doesn't depend on the dimension of feature map.

Dual Objective Revised

- With kernel function $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$

$$\min_{\boldsymbol{\alpha}} \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) - \sum_{i=1}^{N} \alpha_i$$

$$\text{s.t.} \quad \sum_{i=1}^{N} y_i \alpha_i = 0$$

$$0 \leq \alpha_i \leq C \quad \text{for all } i = 1, ..., N$$

- We don't need to evaluate feature map during training.

Kernels - Least Square with Gradient Descent

Kernel method can be applied to any ML algorithms that utilize the inner product between inputs.

For example, a gradient descent step for the least square method is

$$\theta := \theta + \eta \sum_{n=1}^{N} (y_n - \theta^\top \phi(\boldsymbol{x}_n)) \phi(\boldsymbol{x}_n)$$

Note that[1] θ is a linear combination of $\phi(\boldsymbol{x}_1), ..., \phi(\boldsymbol{x}_N)$ if the initial value of $\theta = 0$.

$$\theta = \sum_{n=1}^{N} \alpha_n \phi(\boldsymbol{x}_n)$$

[1] We can prove this inductively.

Kernels - Least Square

The update step can be rewritten as

$$\begin{aligned}\theta &:= \theta + \eta \sum_{n=1}^{N} (y_n - \theta^\top \phi(\boldsymbol{x}_n))\phi(\boldsymbol{x}_n) \\ &= \sum_{n=1}^{N} \alpha_n \phi(\boldsymbol{x}_n) + \eta \sum_{n=1}^{N} (y_n - \theta^\top \phi(\boldsymbol{x}_n))\phi(\boldsymbol{x}_n) \\ &= \sum_{n=1}^{N} \left(\alpha_n + \eta(y_n - \theta^\top \phi(\boldsymbol{x}_n))\right) \phi(\boldsymbol{x}_n)\end{aligned}$$

New α_n depends on the old one via:

$$\alpha_n := \alpha_n + \eta(y_n - \theta^\top \phi(\boldsymbol{x}_n))$$

Kernels - Least Square

From

$$\alpha_n := \alpha_n + \eta(y_n - \theta^\top \phi(\boldsymbol{x}_n))$$

Replacing θ with $\theta = \sum_{n=1}^{N} \alpha_n \phi(\boldsymbol{x}_n)$ gives

$$\alpha_i := \alpha_i + \eta \left(y_n - \sum_{j=1}^{N} \alpha_j \phi(\boldsymbol{x}_j)^\top \phi(\boldsymbol{x}_i) \right)$$

Result in

$$\alpha_i := \alpha_i + \eta \left(y_n - \sum_{j=1}^{N} \alpha_j \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) \right)$$

Properties of Kernels

We don't need to define feature function ϕ explicitly as long as we have a valid kernel function κ.

What are the properties of the valid kernel function then?

Theorem (Mercer)

Let $\kappa : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ be given. Then for κ to be a valid (Mercer) kernel, it is necessary and sufficient that for any $\boldsymbol{x}_1, ..., \boldsymbol{x}_n, (n < \infty)$, the corresponding kernel matrix is symmetric positive semi-definite[2].

From kernel function, we can construct the kernel matrix $\mathbf{K} \in \mathbb{R}^{N \times N}$ filled with $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$.

[2] Kernel matrices are p.s.d

Kernels as Similarity Metrics

Intuitively, if $\phi(\boldsymbol{x}_i)$ and $\phi(\boldsymbol{x}_j)$ are close together, then we might expect $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$ to be large.

Conversely, if $\phi(\boldsymbol{x}_i)$ and $\phi(\boldsymbol{x}_j)$ are nearly orthogonal to each other, then we might expect $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$ to be small.

So, we can think of $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$ as some measurement of how similar are $\phi(\boldsymbol{x}_i)$ and $\phi(\boldsymbol{x}_j)$. For instance,

$$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}{2\sigma^2}\right)$$

This kernel is called Gaussian (Radial Basis Function: RBF) kernel.

Example: Polynomial Kernel

- Polynomial kernel of degree p:

$$\kappa(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^\top \boldsymbol{x}' + 1)^p = \phi(\boldsymbol{x})^\top \phi(\boldsymbol{x}')$$

- If $\boldsymbol{x} = [x_1, x_2]^\top \in \mathbb{R}^2$ and $p = 2$, the corresponding feature map is

$$\phi(\boldsymbol{x}) = [1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2, x_1^2, x_2^2]^\top$$

- Instead of converting $\boldsymbol{x}$ to feature $\phi(\boldsymbol{x})$ and taking inner product,
- It would be much faster to compute the kernel function directly

Example: RBF Kernel

- Radial basis function (RBF) kernel:

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

- The corresponding feature map ϕ has infinite dimensions

$$\phi(x) = \exp(-x^2/2\sigma^2)\left[1, \sqrt{\frac{1}{1!\sigma^2}}x, \sqrt{\frac{1}{2!\sigma^4}}x^2, \sqrt{\frac{1}{3!\sigma^6}}x^3, \ldots\right]$$

- We can still evaluate kernel, but cannot construct feature map.

Additional Examples 1

Linear kernel:

$$\kappa\left(x^{(i)}, x^{(j)}\right) = {x^{(i)}}^{\top} x^{(j)}$$

Squared exponential (Gaussian) kernel:

$$\kappa\left(x^{(i)}, x^{(j)}\right) = \exp\left(-\frac{1}{2}\left(x^{(i)} - x^{(j)}\right)^{\top} \Sigma^{-1}\left(x^{(i)} - x^{(j)}\right)\right)$$

Sigmoid kernel:

$$\kappa\left(x^{(i)}, x^{(j)}\right) = \tanh\left(a \cdot {x^{(i)}}^{\top} x^{(j)} + b\right)$$

Additional Examples 2

Positive scaling:

$$\kappa\left(x^{(i)}, x^{(j)}\right) = c\kappa_1\left(x^{(i)}, x^{(j)}\right)$$

Exponentiation:

$$\kappa\left(x^{(i)}, x^{(j)}\right) = \exp\left(\kappa_1\left(x^{(i)}, x^{(j)}\right)\right)$$

Addition:

$$\kappa\left(x^{(i)}, x^{(j)}\right) = \kappa_1\left(x^{(i)}, x^{(j)}\right) + \kappa_2\left(x^{(i)}, x^{(j)}\right)$$

Multiplication with function:

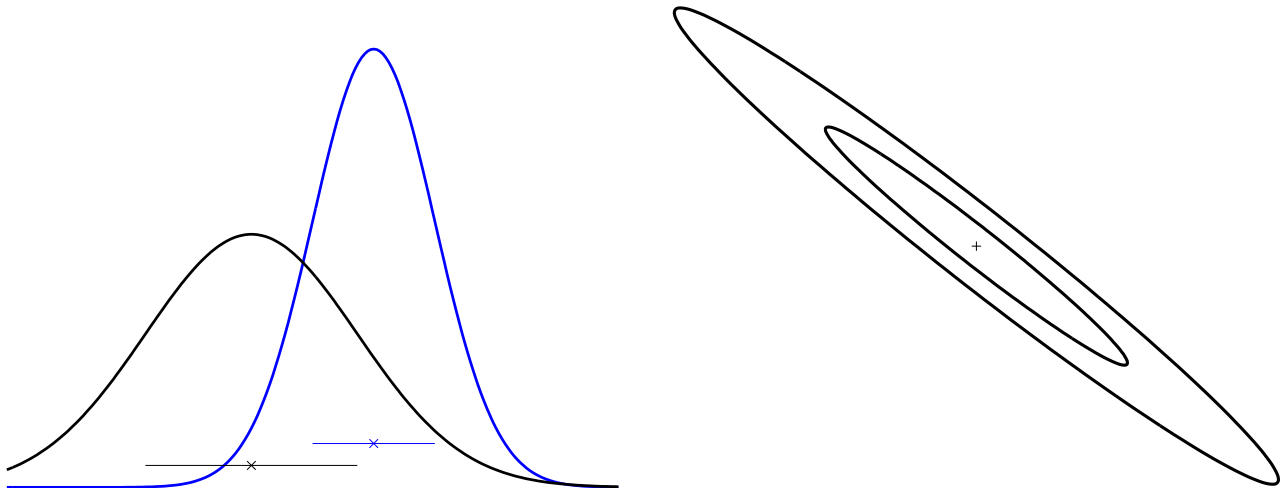
$$\kappa\left(x^{(i)}, x^{(j)}\right) = f\left(x^{(i)}\right)\kappa_1\left(x^{(i)}, x^{(j)}\right)f\left(x^{(j)}\right)$$

Multiplication:

$$\kappa\left(x^{(i)}, x^{(j)}\right) = \kappa_1\left(x^{(i)}, x^{(j)}\right)\kappa_2\left(x^{(i)}, x^{(j)}\right)$$

Table of Contents

The Gaussian Distribution

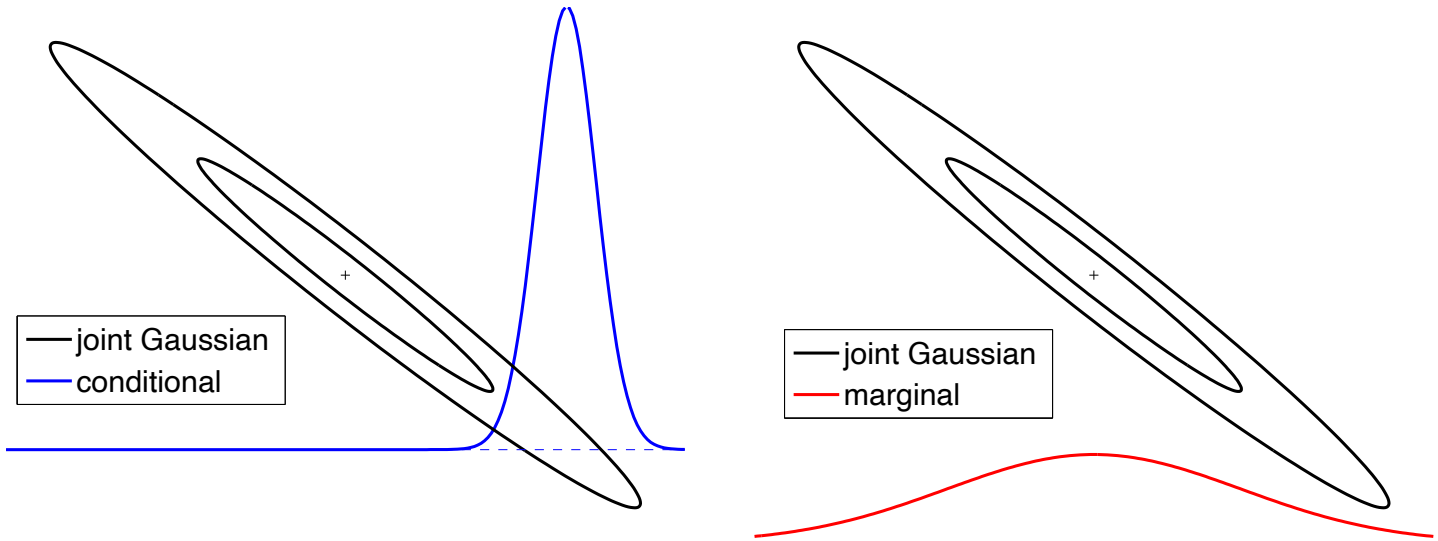


The Gaussian distribution is given by

$$p(\mathbf{x} \mid \mu, \Sigma) = \mathcal{N}(\mu, \Sigma) = (2\pi)^{-D/2}|\Sigma|^{-1/2} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^\top \Sigma^{-1}(\mathbf{x}-\mu)\right)$$

where μ is the mean vector and Σ the covariance matrix.

Conditionals and Marginals of a Gaussian



Both the conditionals and the marginals of a joint Gaussian are again Gaussian.

Gaussian Distribution as a Distribution over Function

- Informally, function $f : \mathcal{X} \to \mathbb{R}$ can be seen as an infinite dimensional vector. For example, if $\mathcal{X} = \mathbb{R}$, we can think

$$f(x) = [f(x = 1), f(x = 2), f(x = 3), ...]$$

- Given dataset $D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$, if the collection of outputs $\boldsymbol{y} = (y_1, y_2, ..., y_n)$ are drawn from Gaussian distribution

$$\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$$

  We say the Gaussian distribution defines a distribution over function outputs.

Gaussian Distribution as a Distribution over Function

- Informally, function $f : \mathcal{X} \to \mathbb{R}$ can be seen as an infinite dimensional vector. For example, if $\mathcal{X} = \mathbb{R}$, we can think

$$f(x) = [f(x = 1), f(x = 2), f(x = 3), ...]$$

- Given dataset $D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$, if the collection of outputs $\boldsymbol{y} = (y_1, y_2, ..., y_n)$ are drawn from Gaussian distribution

$$\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$$

  We say the Gaussian distribution defines a distribution over function outputs.

- But where are the inputs $(x_1, x_2, ..., x_n)$?
- And how can we predict output given new input x_*?

Kernel as Covariance Matrix

Note that if we define a Gaussian over the collection of outputs, the distribution is defined without inputs.

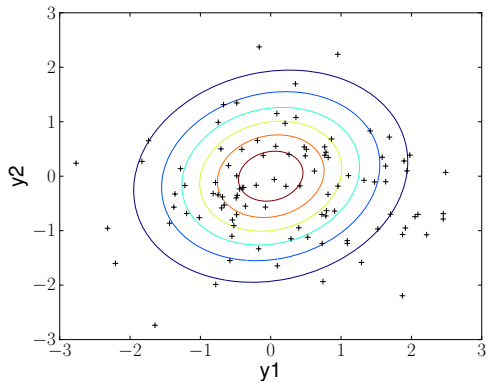
$$\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$$

- Idea: construct the covariance matrix using a kernel function $\kappa(x, x')$!

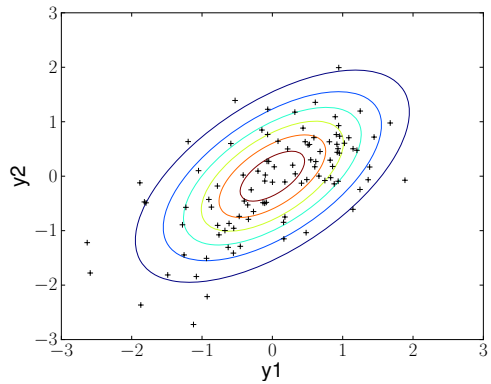
$$\Sigma_{i,j} = \mathbf{K}_{i,j} = \kappa(x_i, x_j)$$

- Why:
  1. Smoothness assumption: if two inputs x_1 and x_2 are close, i.e. $x_1 \approx x_2$, then their outputs are close, i.e. $f(x_1) \approx f(x_2)$.
  2. The covariance measures the correlation between two dimensions.
  3. If $\mathbf{K}_{i,j}$ large, then y_i and y_j are likely to be similar.
  4. If $\mathbf{K}_{i,j}$ small, then y_i and y_j are likely to be uncorrelated.

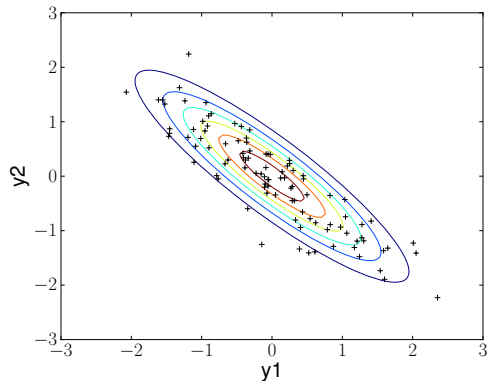
A 2D Gaussian: Varying the Covariance



$$\mathbf{K} = \begin{bmatrix} 1 & 0.14 \\ 0.14 & 1 \end{bmatrix}$$



$$\mathbf{K} = \begin{bmatrix} 1 & 0.6 \\ 0.6 & 1 \end{bmatrix}$$



$$\mathbf{K} = \begin{bmatrix} 1 & -0.9 \\ -0.9 & 1 \end{bmatrix}$$

Putting Them Together

- Given dataset $D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$, we can define a probability density function over the outputs:

$$p\left(\begin{bmatrix} f(x_1) \\ \vdots \\ f(x_n) \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} \mu(x_1) \\ \vdots \\ \mu(x_n) \end{bmatrix}, \begin{bmatrix} \kappa(x_1, x_1) & \dots & \kappa(x_1, x_n) \\ \vdots & \ddots & \vdots \\ \kappa(x_n, x_1) & \dots & \kappa(x_n, x_n) \end{bmatrix}\right)$$

- Often we assume $\mu(x) = 0$, $\forall x$ as a prior of function.
- We can define a distribution over functions!

Putting Them Together

- Given dataset $D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$, we can define a probability density function over the outputs:

$$p\left(\begin{bmatrix} f(x_1) \\ \vdots \\ f(x_n) \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} \mu(x_1) \\ \vdots \\ \mu(x_n) \end{bmatrix}, \begin{bmatrix} \kappa(x_1, x_1) & \dots & \kappa(x_1, x_n) \\ \vdots & \ddots & \vdots \\ \kappa(x_n, x_1) & \dots & \kappa(x_n, x_n) \end{bmatrix}\right)$$

- Often we assume $\mu(x) = 0$, $\forall x$ as a prior of function.
- We can define a distribution over functions!
- How can we predict output given new input x_*?
  - Do you remember Gaussian identity? (from linear regression)

Lemma (Gaussian identities[3])

Let the joint distribution of $z = [x^\top, y^\top]^\top$ *be*

$$z = \begin{bmatrix} x \\ y \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} a \\ b \end{bmatrix}, \begin{bmatrix} A & C \\ C^\top & B \end{bmatrix} \right) \quad \text{with cross-covariance } C = \mathbb{E}[xy^\top] \,,$$

so that $x \sim \mathcal{N}(a, A)$ *and* $y \sim \mathcal{N}(b, B)$*. Then, the conditional distributions are given as:*

$$p(x \mid y) = \mathcal{N}(x \mid a + CB^{-1}(y - b), A - CB^{-1}C^\top) \,,$$
$$p(y \mid x) = \mathcal{N}(y \mid b + C^\top A^{-1}(x - a), B - C^\top A^{-1}C) \,.$$

[3] c.f. Pattern Recognition and Machine Learning, Sec 2.3.1

Prediction with Gaussian Identity

Let x_* be the test input and y_* be the predicted value. The joint distribution of $\boldsymbol{y}$ and y_* given D is

$$p\left(\boldsymbol{y}, y_* \mid \boldsymbol{x}, \boldsymbol{x}^*\right) = \mathcal{N}\left(\begin{pmatrix} \boldsymbol{y} \\ y_* \end{pmatrix} \middle| \mathbf{0}, \begin{bmatrix} \mathbf{K}_{\mathbf{x},\mathbf{x}} & \mathbf{K}_{\mathbf{x},x_*} \\ \mathbf{K}_{x_*,\mathbf{x}} & \mathbf{K}_{x_*,x_*} \end{bmatrix}\right)$$

where $\mathbf{K}_{x_*,\mathbf{x}}$ is

$$\mathbf{K}_{x_*,\mathbf{x}} = [\kappa(x_*, x_1), ..., \kappa(x_*, x_n)]$$

Prediction with Gaussian Identity

Let x_* be the test input and y_* be the predicted value. The joint distribution of $\boldsymbol{y}$ and y_* given D is

$$p\left(\boldsymbol{y}, y_* \mid \boldsymbol{x}, \boldsymbol{x}^*\right) = \mathcal{N}\left(\begin{pmatrix} \boldsymbol{y} \\ y_* \end{pmatrix} \middle| \mathbf{0}, \begin{bmatrix} \mathbf{K}_{\mathbf{x},\mathbf{x}} & \mathbf{K}_{\mathbf{x},x_*} \\ \mathbf{K}_{x_*,\mathbf{x}} & \mathbf{K}_{x_*,x_*} \end{bmatrix}\right)$$

where $\mathbf{K}_{x_*,\mathbf{x}}$ is

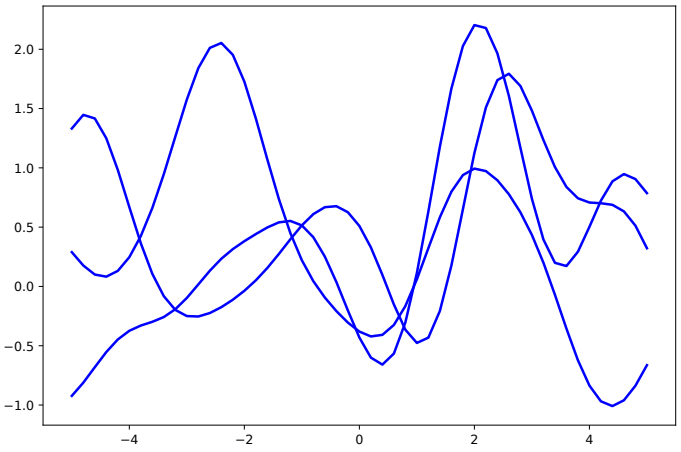
$$\mathbf{K}_{x_*,\mathbf{x}} = [\kappa(x_*, x_1), ..., \kappa(x_*, x_n)]$$

With Gaussian identity

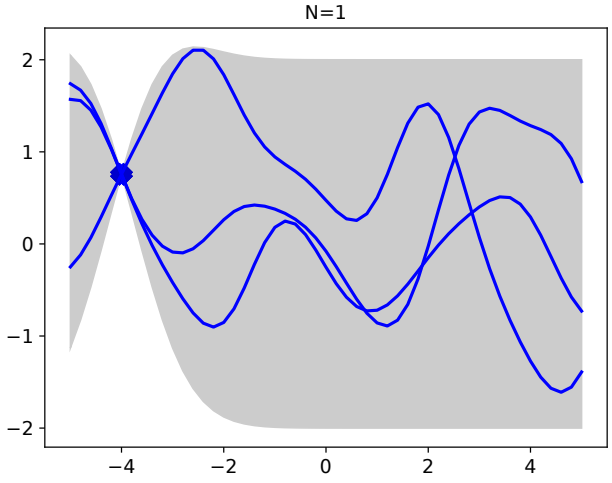
$$\begin{aligned} p\left(y_* \mid \mathbf{x}_*, \mathcal{D}\right) &= \mathcal{N}\left(y_* \mid \boldsymbol{\mu}_*, \Sigma_*\right) \\ \boldsymbol{\mu}_* &= \mathbf{0} + \mathbf{K}_{\boldsymbol{x},x_*}^\top \mathbf{K}_{\boldsymbol{x},\boldsymbol{x}}^{-1}(\boldsymbol{y} + \mathbf{0}) \\ \Sigma_* &= \mathbf{K}_{x_*,x_*} - \mathbf{K}_{\boldsymbol{x},x_*}^\top \mathbf{K}_{\boldsymbol{x},\boldsymbol{x}}^{-1}\mathbf{K}_{\boldsymbol{x},x_*} \end{aligned}$$

We can generalize this to multiple predicted values $\boldsymbol{y}_*$ given $\boldsymbol{x}_*$

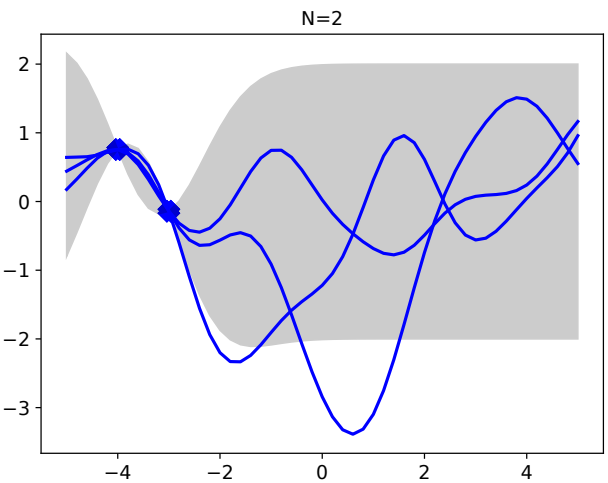
Examples: Prediction with Observations



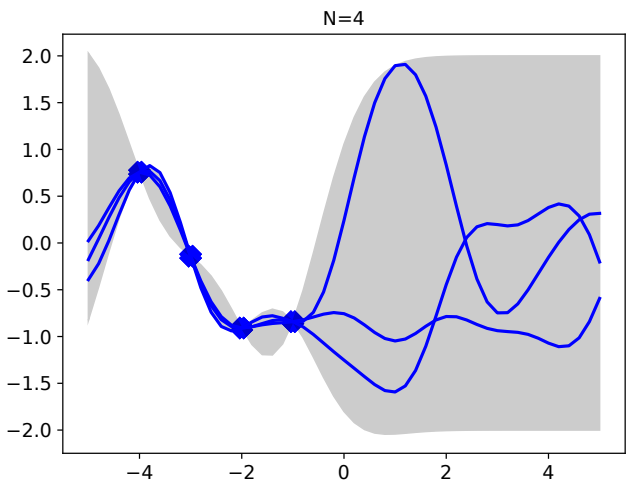
(a)



(b)



(c)



(d)

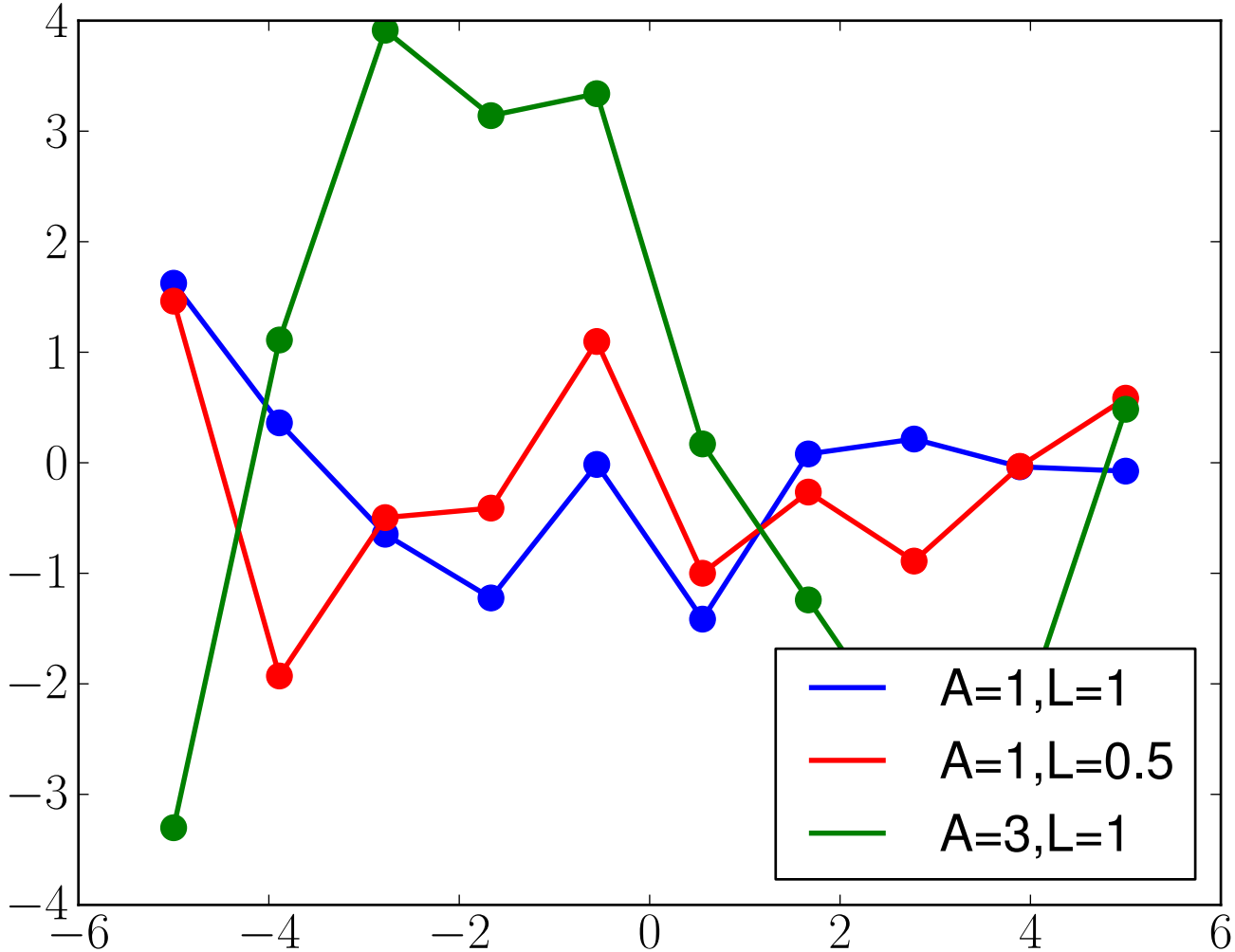
What does it mean by training Gaussian Process?

A commonly used kernel function to construct covariance is the squared exponential:

$$\kappa_{\mathrm{SE}}\left(x_i, x_j, ; A, L\right) = A^2 \exp\left\{-0.5 \cdot \frac{\left(x_i - x_j\right)^2}{L^2}\right\}$$

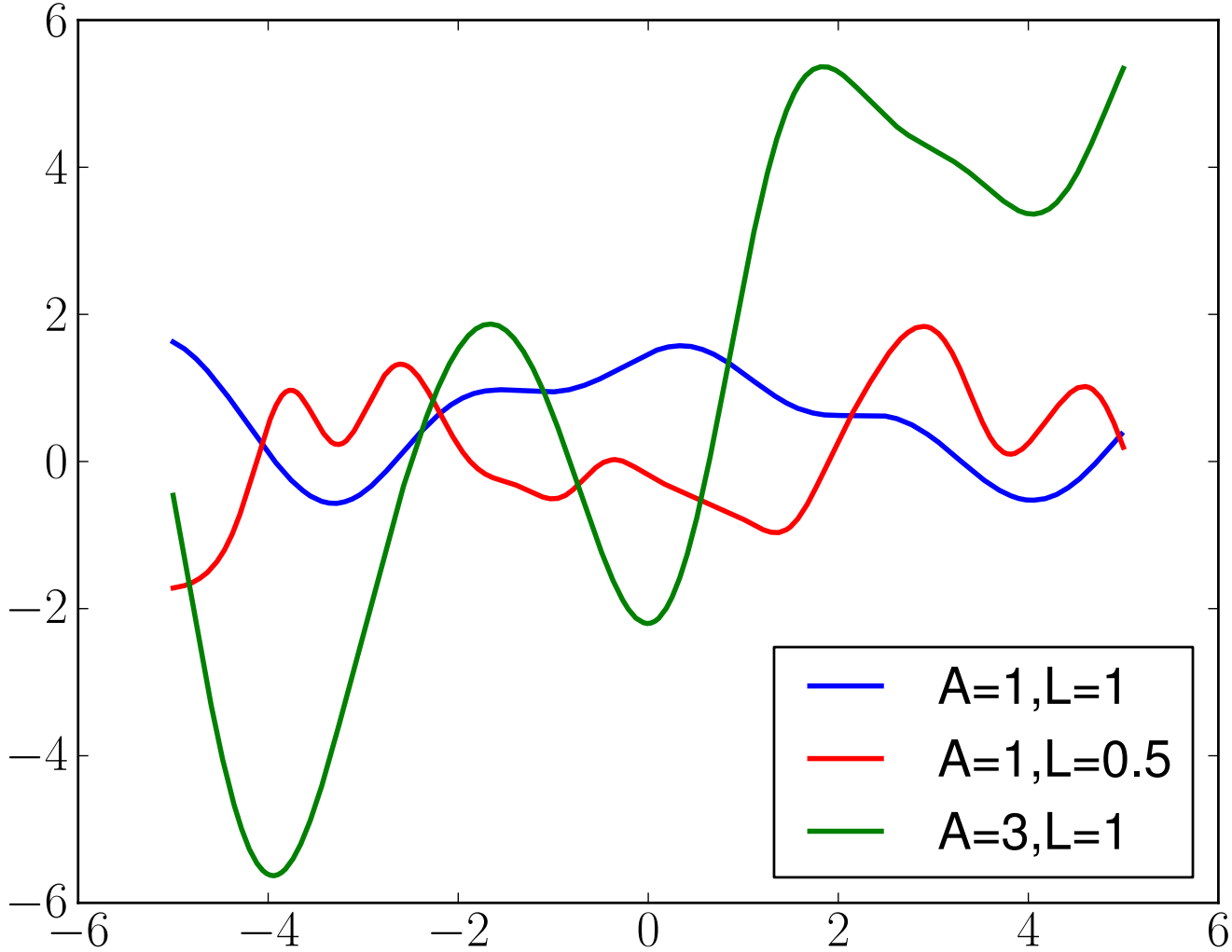
- A: Overall correlation, amplitude
- L: Scaling parameter, smoothness
- Hyperparameters can be tuned to maximize some objective (MLE, MAP)
- GP aims to find the best kernel function and their hyper-parameters

Examples: Sampled Functions



10D Gaussian

Examples: Sampled Functions



500D Gaussian

Noisy Gaussian Process Regression

- So far, we have assume that the observed outputs $\boldsymbol{y}$ do not contain noise.
- As in linear regression, it is more natural to add observation noise into GP:

$$y = f(x) + \epsilon$$

where $f \sim \mathrm{GP}\left(m(x), k\left(x, x'\right)\right)$ and $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$.

  - GP: prior of f
  - $\mathcal{N}(f(x), \sigma_\epsilon^2)$: likelihood of y
  - We can compute posterior exactly! - Check the textbook

Further Readings

- Textbook: Chapter 17.1 (Mercer kernels)
- Textbook: Chapter 17.2 (Gaussian Processes)
- Properties of kernels (on PLMS)