Probabilistic Deep Learning:

1. Kernel Methods & Gaussian Process

ESC 2024 Spring Session 1주차

Contents

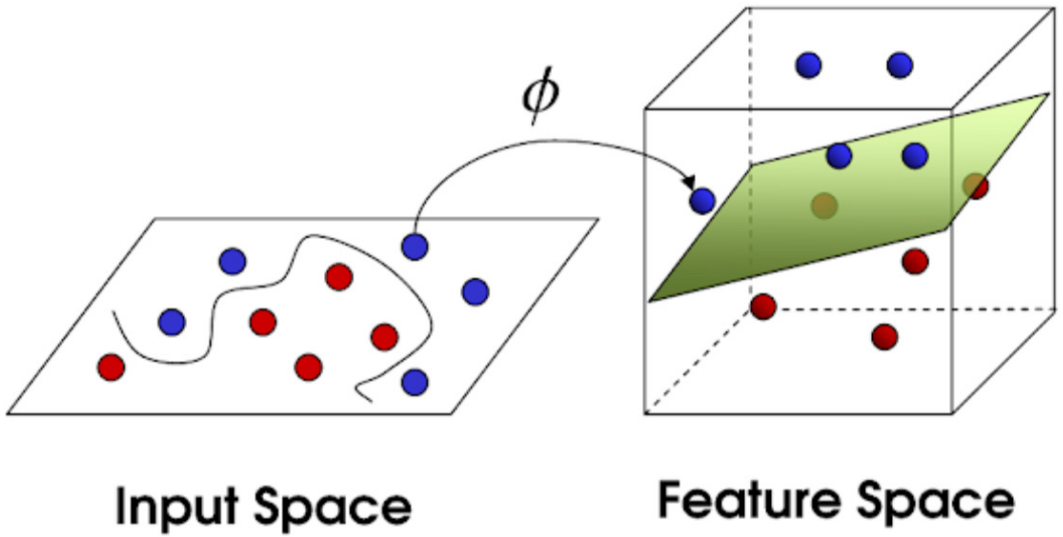
1

Dual Representations

Kernel methods

Introduction

본 내용으로 들어가기에 앞서, Basis Function(혹은 Feature map)에 대한 간단한 이해와 복습이 필요하다.

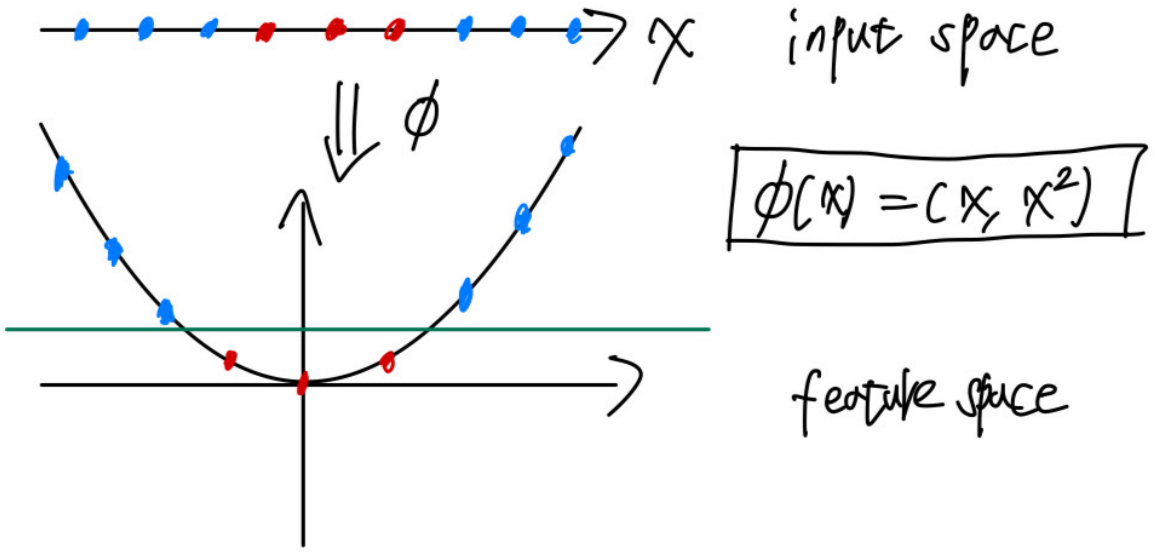


왼쪽의 Input Space $\boldsymbol{X}$에서 빨간 점과 파란 점을 classify하는 모델을 만들고 싶다. 하지만, linear한 모델 $f(\boldsymbol{X})$로는 이 문제를 해결할 수 없다.

이때, non-linear한 모델을 사용할 수도 있겠지만 input space $\boldsymbol{X}$를 linear model이 classify할 수 있는 feature space $\emptyset(\boldsymbol{X})$로 바꾸면 복잡한 non-linear 모델을 사용하지 않아도 된다.
이때, Input Space $\boldsymbol{X}$를 feature space $\emptyset(\boldsymbol{X})$로 mapping해주는 함수를 Basis function(=Feature map)이라고 한다.

다음 페이지에서 예시를 하나만 더 살펴보자.

Introduction



위와 같은 Input space에서는 linear한 모델로는 classification이 불가능하다. 하지만, feature space를 아래와 같이 설정한다면 linear model로도 해결이 가능해진다.

이때, Basis function $\emptyset(x) = (x, x^2)$이다.

하지만, 실제 문제 상황은 위와 같이 단순하지 않기에

1. Feature map $\emptyset(\boldsymbol{X})$를 찾는 것 자체가 매우 어렵고
2. 찾은 후에도 계산이 쉽지 않다.

이러한 문제 상황에서 고안된 것이 바로 kernel인데, 이 흐름을 가지고 다음 내용들을 보도록 하자.

기본적으로, Kernel Function은 아래와 같이 정의된다.

$$k(\boldsymbol{x_1}, \boldsymbol{x_2}) = \emptyset(\boldsymbol{x_1})^T \emptyset(\boldsymbol{x_2})$$

즉, kernel function은 feature space에서의 내적으로 이해할 수 있다.

$\emptyset$를 정의하는 것에 따라 수없이 다양한 kernel을 만들 수 있다.

가장 간단하게, $\emptyset(\boldsymbol{X}) = \boldsymbol{X}$ 로 정의한다면, $k(\boldsymbol{x_1}, \boldsymbol{x_2}) = \boldsymbol{x_1}^T \boldsymbol{x_2}$이다.

이것은 linear kernel이라고 하는데, 앞으로 이것 말고도 다양한 kernel들을 살펴볼 것이다.

Dual Representations

여기서는 Dual Representations에 알아볼 것인데,

말 그대로 두 개의 표현식이 있다는 의미이다. 무엇을 두 가지로 표현할까?

한 예시로, 굉장히 익숙한 아래의 식을 보자.

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

이때, $\boldsymbol{w}$는 파라미터, $\boldsymbol{x_n}$은 Input, t_n은 output으로 notation이 살짝 다르긴 하지만 Ridge Regerssion의 loss function이다.

이때, $\boldsymbol{w}$**를** 구하기 위해 식을 $\boldsymbol{w}$**에** 대해 미분하고 =0으로 두어 방정식을 풀면

$$\mathbf{w} = -\frac{1}{\lambda}\sum_{n=1}^{N}\{\mathbf{w}^{\mathbf{T}}\phi(\mathbf{x_n}) - t_n\}\phi(\mathbf{x_n})$$

식을 간단히 하기 위해

$a_n = -\frac{1}{\lambda}\{\boldsymbol{w^T}\phi(\boldsymbol{x_n}) - t_n\}$**으로** 정의하고, $\boldsymbol{a}$**를** a_n**을** 원소로 가지는 N x 1 열벡터로 정의하자. 즉, $\boldsymbol{a} = (a_1, \ldots, a_N)^T$이다.

또, $\boldsymbol{\Phi}$**를** n번째 행이 $\phi(\boldsymbol{x_n})^T$인 design matrix라고 하자. Then,

$$(1) = \sum_{n=1}^{N} a_n\phi(\mathbf{x_n}) = \boldsymbol{\Phi}^{\mathbf{T}}\mathbf{a}$$

이제, 이 $\boldsymbol{w} = \boldsymbol{\Phi^T a}$**를** $\boldsymbol{J(w)}$에 대입하면,

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

이것을 $\boldsymbol{a}$에 대한 함수로 보아 $\boldsymbol{J(a)}$**로** 두면

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

여기서, 원래 파라미터 벡터 $\boldsymbol{w}$가 아닌 새로운 파라미터 벡터 $\boldsymbol{a}$에 대해 loss function을 표현했으므로, 이것을 **Dual Representation**이라고 한다.

Dual Representations

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

결과적으로 이 식을 도출했는데, 딱 봐도 식이 매우 지저분하다.

식을 간단히 하기 위해 Gram matrix $\boldsymbol{K}$를 정의하는데,

$\boldsymbol{K} = \boldsymbol{\Phi}^{\boldsymbol{T}}\boldsymbol{\Phi}$이다. $K_{nm} = \phi(\boldsymbol{x_n})^T\phi(\boldsymbol{x_m})$인데, 여기서 아까 정의했던

Kernel function을 떠올리면, $K_{nm} = \phi(\boldsymbol{x_n})^T\phi(\boldsymbol{x_m}) = k(\boldsymbol{x_n}, \boldsymbol{x_m})$이다.

Gram matrix를 통해 위의 지저분한 식을 깔끔히 정리하면

$$J(\mathbf{a}) = \frac{1}{2}\mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{K}\mathbf{a} - \mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{t} + \frac{1}{2}\mathbf{t}^{\mathrm{T}}\mathbf{t} + \frac{\lambda}{2}\mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{a}$$

이 식을 다시 한 번 $\boldsymbol{a}$에 대해 미분하고 0으로 두어 방정식을 풀면

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

자, 이제 다시 처음의 linear regression 모델로 돌아와보자.

$\boldsymbol{x}$라는 새로운 data point가 주어졌을 때, 여기서의 예측값

$y(\boldsymbol{x}) = \boldsymbol{w}^{\boldsymbol{T}}\phi(\boldsymbol{x})$ 이다. 여기에 아까 구한 $\boldsymbol{w} = \boldsymbol{\Phi}^{\boldsymbol{T}}\boldsymbol{a}$를 대입하면

$y(\boldsymbol{x}) = \boldsymbol{w}^{\boldsymbol{T}}\phi(\boldsymbol{x}) = \boldsymbol{a}^{\boldsymbol{T}}\boldsymbol{\Phi}\phi(\boldsymbol{x})$이다. 여기에 아까 구한

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

를 대입하면

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

이때, $\boldsymbol{K}(\boldsymbol{x})$는 n번째 항 $K_n(\boldsymbol{x})$가 $k(\boldsymbol{x_n}, \boldsymbol{x})$ 인 n x 1 열벡터이다.

Dual Representations

지금까지 열심히 Dual Representation 식을 정리해왔는데, 이렇게 정리하는 의의가 뭘까?

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

식을 자세히 살펴보면, Dual Representation으로 표현한 결과 least square estimator에 $\phi(x)$**에** 대한 식이 없는 것을 확인할 수 있다.

즉, 구하기 어려운 Basis Function 대신 kernel function만을 이용해 least square estimator를 나타낼 수 있다.

주목할 point가 한 가지 더 있다.

그동안 우리가 배운 linear regression model $y(x) = w^T\phi(x)$ 에서, Training data를 통해 $\hat{w}$**을** 학습하면, 새로운 data point x_{new}에서의 예측값 $y_{new} = \hat{w}\phi(x_{new})$였다.

즉, training data는 파라미터를 학습할 때만 쓰이고, 예측 시에는 쓰이지 않았다.

반면, Daul Representation으로 구한 식을 통해 예측하면

$$y_{new} = \mathbf{k}(\mathbf{x_{new}})^\mathbf{T}(\mathbf{K} + \lambda\mathbf{I}_N)^{-1}\mathbf{t}$$

이다. 식을 자세히 살펴보면, **예측 과정에서도 Training data K가 쓰이고 있는 것**을 확인할 수 있다.

이러한 방법을 **memory-based methods**라고 한다.

Dual Representations

사실, Dual Representation으로 계산하는 것이 만능은 아니다.
원래 형태에서는 parameter를 estimate할 때 M x M matrix를 Inverse하는 연산을 해야 했다.
(예를 들어, LSE = $(\boldsymbol{x}'\boldsymbol{x})^{-1}\boldsymbol{x}'\boldsymbol{y}$ 에서 $(\boldsymbol{x}'\boldsymbol{x})^{-1}$는 M x M matrix 이다)
반면, Dual Representation으로 바꾸면 N x N matrix를 inverse 해야 한다.
($\boldsymbol{a} = (\boldsymbol{K} + \lambda\boldsymbol{I}_n)^{-1}\boldsymbol{t}$ 에서 $\boldsymbol{I}_n$을 inverse 해야 한다)

M은 변수의 개수, N은 data의 수 이므로 보통 상황에선 N이 M보다 훨씬 크다. 즉, 계산이 훨씬 복잡해지게 된다.

그럼에도 이를 사용하는 이유는, Basis Function을 사용하지 않고 kernel function에 대한 식으로만 parameter를 estimate할 수 있기 때문이다.
도입부에서 Basis Function $\phi(\boldsymbol{x})$을 찾는게 매우 어렵다고 했던 것을 기억하자.

Dual Representation을 사용하면 least square를 구할 때 $\phi(\boldsymbol{x})$를 사용하지 않기 때문에, feature space를 보다 높은 차원으로 복잡하게 설정할 수 있게 된다. 이것이 우리가 Dual Representation을 사용하는 이유이다.

2

Constructing Kernels

Kernel methods

Constructing Kernels

앞에서 배운 내용들을 활용하려면 결국 kernel function이 필요한데, 아무 식이나 가져다 kernel로 쓸 수 없고 '**valid**'한 kernel이 필요하다.

그럼, 주어진 kernel이 'valid'한지 어떻게 알 수 있을까?

한 방법은 basis function $\phi(\boldsymbol{x})$를 직접 찾는 것이다.

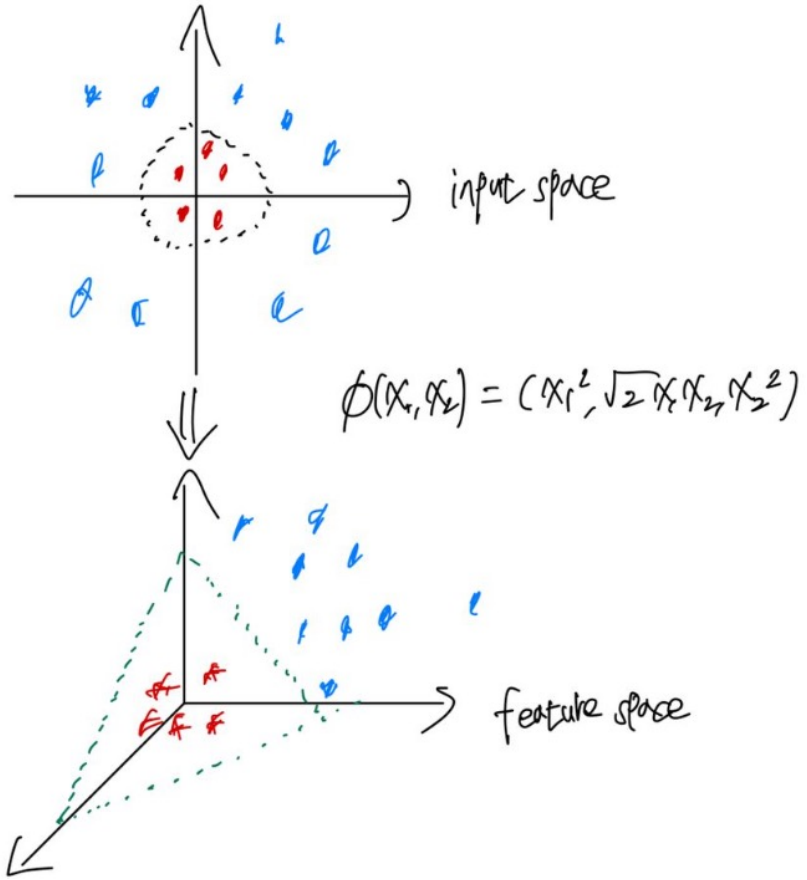
예를 들어, kernel $k(\boldsymbol{x}, \boldsymbol{z}) = (\boldsymbol{x}^T \boldsymbol{z})^2$이 valid한 kernel일까?

Dimension을 2차원이라고 가정하면, $x = (x_1, x_2), z = (z_1, z_2)$이고,

$$
\begin{aligned}
k(\mathbf{x}, \mathbf{z}) &= (\mathbf{x}^{\mathbf{T}} \mathbf{z})^2 = (x_1 z_1 + x_2 z_2)^2 \\
&= x_1^2 z_1^2 + 2 x_1 z_1 x_2 z_2 + x_2^2 z_2^2 \\
&= (x_1^2, \sqrt{2} x_1 x_2, x_2^2)(z_1^2, \sqrt{2} z_1 z_2, z_2^2)^T \\
&= \phi(\mathbf{x})^T \phi(\mathbf{z})
\end{aligned}
$$

여기서 우리가 찾은 map을 찾을 수 있으므로 이것은 valid한 kernel임을 확인할 수 있다.

tmi) 사실 앞에서 예시로 든 basis function은 아래의 상황을 나타내는 것이긴 하다



Constructing Kernels

But, 계속해서 나오지만, 실제 복잡한 문제에서는 Basis Function $\phi(x)$를 직접 찾기가 어렵다. Basis Function을 찾지 않고도 kernel이 valid한지 확인하는 방법은 없을까??

Theorem. 어떤 kernel이 valid하기 위한 필요충분 조건은, Gram matrix K가 모든 가능한 $\{x_n\}$에 대해 positive semi-definite인 것이다.

여기에 더해, valid한 두 kernel이 있다면, 아래의 kernel들도 모두 valid한 커널들이다.

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.

Constructing Kernels

이를 이용해서 이미 증명된 여러 kernel들이 있고, 우리는 필요에 따라 이것들을 적절히 선택하여 사용하면 된다.
이들 중 대표적인 몇 가지를 살펴보고, valid한 kernel인지 확인해보자.

1. Linear kernel: 초반부에 정의한 가장 간단한 kernel이다.

$$k(\mathbf{x_1}, \mathbf{x_2}) = \mathbf{x_1^T x_2}$$

2. Polynomial kernel: Linear kernel이 확장된 형태로, 앞서 소개한 정리를 활용하여 valid한 kernel이라는 것을 확인할 수 있다.

$$k(\mathbf{x_1}, \mathbf{x_2}) = (\mathbf{x_1^T x_2} + c)^M$$

3. Gaussian kernel:

$$k(\mathbf{x_1}, \mathbf{x_2}) = exp(-||\mathbf{x_1} - \mathbf{x_2}||^2/2\sigma^2)$$

다음 과정을 통해 Linear kernel들의 곱 형태로 표현하면 valid kernel이라는 것을 확인할 수 있다.

$$||\mathbf{x_1} - \mathbf{x_2}||^2 = \mathbf{x_1^T x_1} + \mathbf{x_2^T x_2} - \mathbf{2x_1^T x_2}$$

$$k(\mathbf{x_1}, \mathbf{x_2}) = exp(-\mathbf{x_1^T x_1}/2\sigma^2)exp(\mathbf{x_1^T x_2}/\sigma^2)exp(-\mathbf{x_2^T x_2}/2\sigma^2)$$

4. Sigmoid kernel:

$$k(\mathbf{x_1}, \mathbf{x_2}) = tanh\{a(\mathbf{x_1^T x_2}) + b\}, \quad (a, b > 0)$$

이 밖에도 수많은 다양한 kernel들이 있다.

3

Radial basis Function Networks

Kernel methods

Radial basis Function Networks

이번 절에서는 Basis Function의 한 형태인 Radial Basis Function에 대해 소개하고, 이를 이용한 간단한 kernel regression모델인 *Nadaraya-Watson* model에 대해 알아볼 것이다.

Radial Basis Function이란 중심 $\boldsymbol{\mu_j}$로부터의 radial distance에만 의존하는 함수를 의미한다.
즉, $\phi_j(\boldsymbol{x}) = h(||\boldsymbol{x} - \boldsymbol{\mu_j}||)$이고, 이때 함수 h로는 여러가지를 적용할 수 있지만, 대표적으로 Gaussian Function등을 적용해 볼 수 있을 것이다.

이와 같은 Radial Basis Function을 사용하여 kernel regerssion (또는 kernel smoothing)을 수행할 수 있다.
Kernel regerssion이란 weighted average 를 이용하는데, 결국 우리가 예측하고자 하는 data point와 가까운 값들에게 weight를 더 주겠다는 것이다. 이때, 가까운 정도를 Radial Basis Function을 이용한 kernel을 통해 계산하게 된다.

먼저, weighted average의 개념을 알아야 한다. $(y_1, \dots, y_n)$까지 데이터가 있다고 할 때, 평균은 $\frac{1}{n}\sum_{i=1}^{n}$로 계산되는 반면, 가중평균은 다음과 같이 계산된다.

$$\frac{\sum_{i=1}^{n} w_i y_y}{\sum_{i=1}^{n} w_i}$$

이때, w_i는 각 y_i들의 weight로, 평균을 계산할 때 얼마만큼의 비중을 가지는지 결정한다.

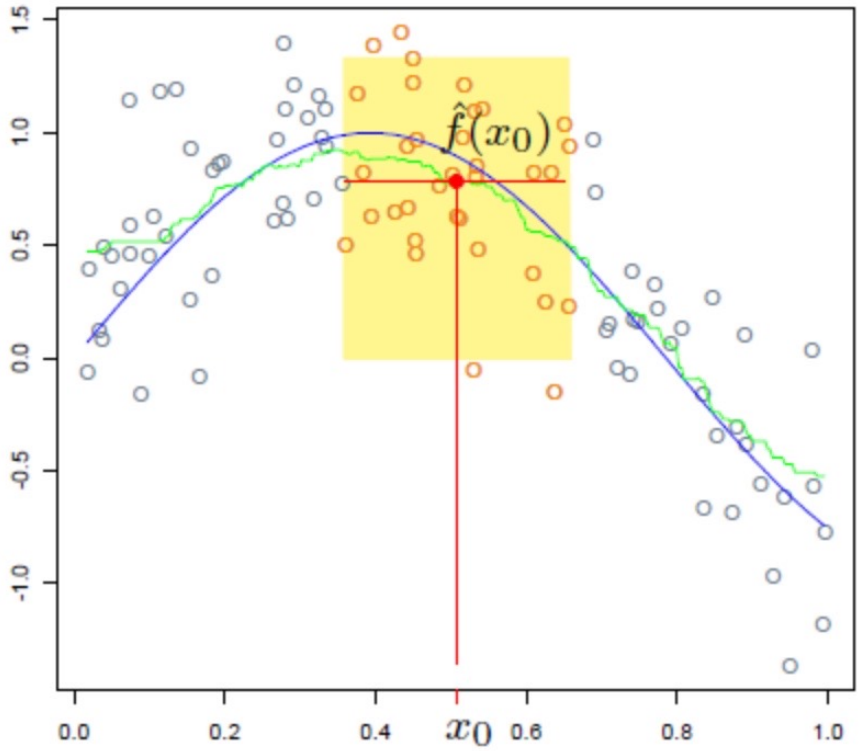
Radial basis Function Networks

Nadaraya-Watson model에 대해 알아보기 전에, 가까운 point들을 이용하여 함수를 fitting하는 *K*-nearest Neighbor Average 방법에 대해 알아보자.

$x = x_0$에서의 *K*-nearest Neighbor Average값은 다음과 같이 정의된다.

$$\hat{f}(x_0) = \frac{1}{k} \sum_{x_i \in N_k(x_0)} y_i$$

이때, $N_k(x_0)$는 training set의 data중, x_0**로**부터 가장 가까운 *K*개 점들의 집합을 의미한다. 즉, *K*-nearest Neighbor Average Method에서는 x_0**로**부터 가장 가까운 *K*개 점들의 y값을 모두 동일한 weight로 평균내어 fitted value를 구하게 된다.



K-nearest Neighbor Average Method의 예시로, 초록색 선이 Fitted line이다. 주황색 점들이 $\hat{f}(x_0)$의 계산에 영향을 준 점들인데, 이 방법에서는 모두 동일한 비중으로 사용되었다.
그러나, fitted line을 보면, 울퉁불퉁하고 불연속점이 있다는 특징이 있다. 이는 가장 가까운 *K*개 점의 집합에서 점들이 나가고 들어오는 과정에서 불연속이 생기기 때문이다.

Radial basis Function Networks

K-nearest Neighbor Average Method는 가까운 k개의 점들을 구한 후, 이들을 모두 **동일한** weight로 average했기 때문에 fitted line에 불연속이 생기고 울퉁불퉁해지는 문제가 발생했다.

만약, 가까운 점일수록 연속적으로 **더 많은** weight를 준다면, 예측의 성능도 좋아지고 fitted line의 불연속 문제도 해결할 수 있지 않을까? 이것이 *Nadaraya-Watson* model의 아이디어이다.

앞의 weighted average를 떠올려 보자.

$$\frac{\sum_{i=1}^{n} w_i y_y}{\sum_{i=1}^{n} w_i}$$

Nadaraya-Watson model에서는 weight w_i값으로 $\frac{K_\lambda(x_0, x_i)}{\sum_{i=1}^{n} K_\lambda(x_0, x_i)}$ 를 사용한다.

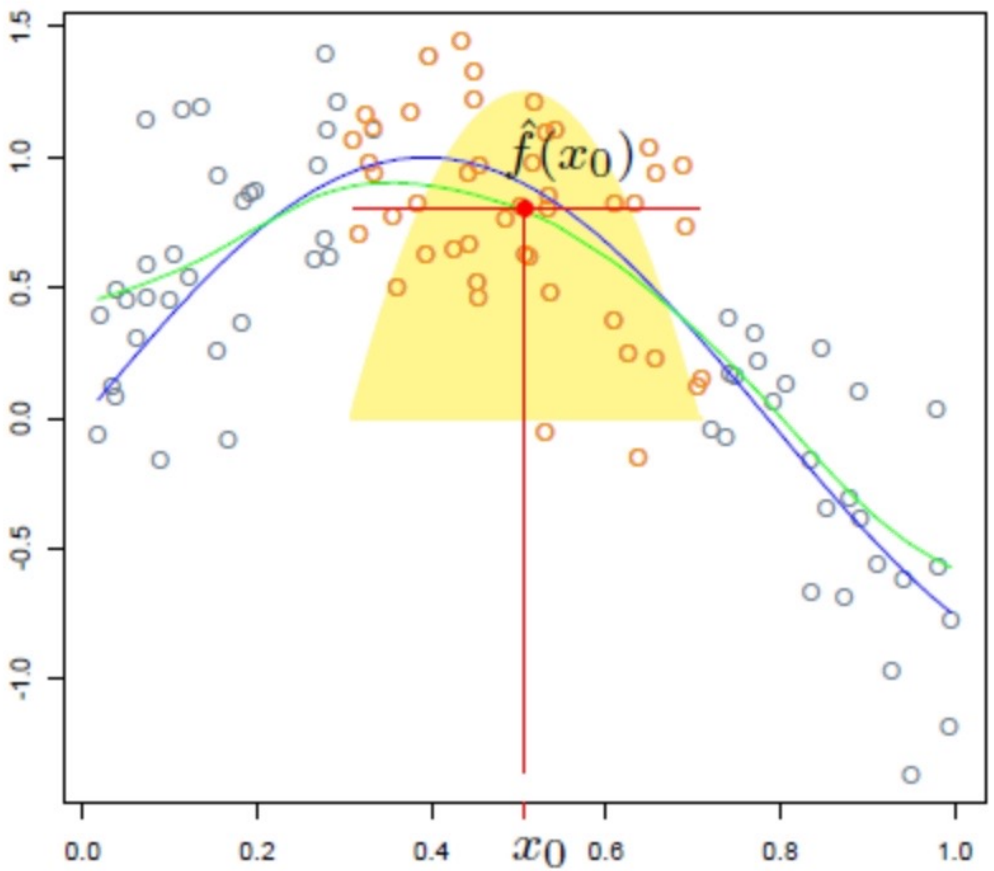
여기서, $K_\lambda(x_0, x_i)$는 x_i와 x_0 가 가까울수록 큰 값을 가지는 kernel로, 역시 대표적으로 Gaussian Kernel을 사용할 수 있다.

즉, x_i와 x_0 가 가까울수록 y_i는 더 큰 weight 가지게 된다.

또한, 이 weight들을 모두 더하면 1이므로, 결과적으로 $x = x_0$에서의 *Nadaraya-Watson* model의 fitted value는 다음과 같다.

$$\hat{f}(x_0) = \sum_{i=1}^{n} \frac{K_\lambda(x_0, x_i)}{\sum_{i=1}^{n} K_\lambda(x_0, x_i)} y_i$$

Radial basis Function Networks



Nadaraya-Watson model로 만든 fitted line이다.

Weight에서 사용된 kernel $K_\lambda(x_0, x_i)$**는** 연속 함수이기 때문에 weight 또한 연속적이게 되고, *K*-nearest Neighbor Average를 사용할 때와는 달리 fitted line이 smooth해진 것을 확인할 수 있다.

그렇기에 이렇게 kernel을 사용한 regression을 kernel smoothing이라고도 한다.

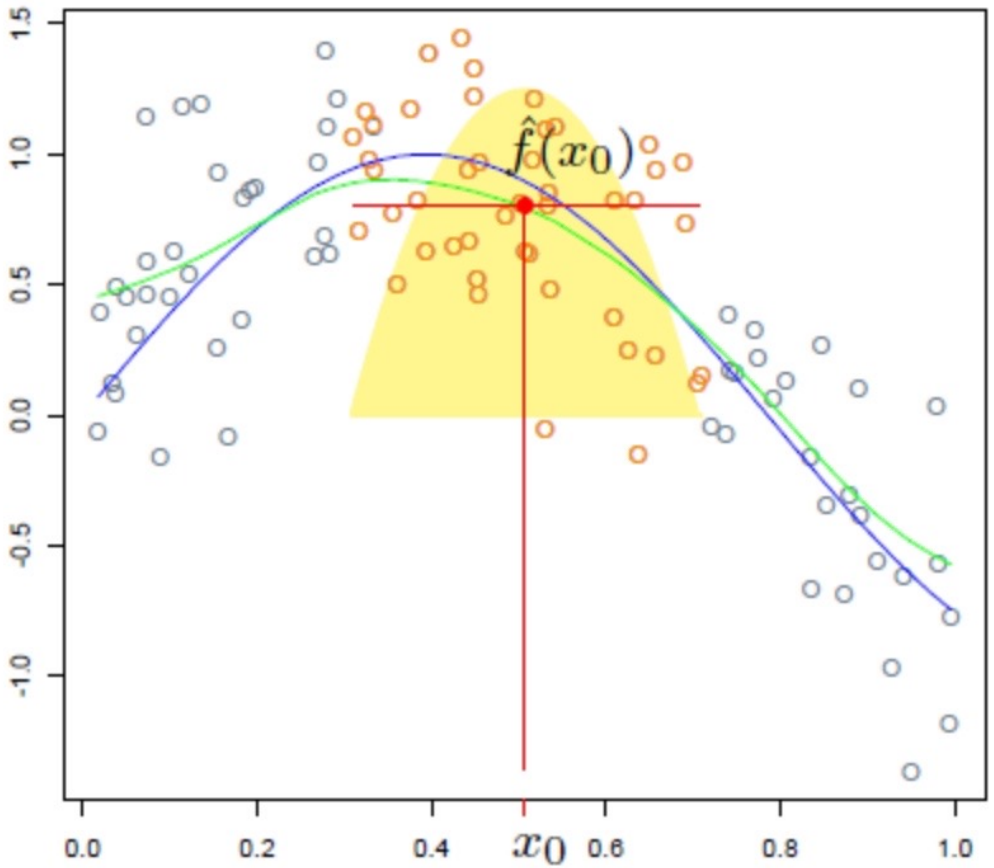
$K_\lambda(x_0, x_i)$**에**서 λ**는** kernel의 표준편차로, λ가 커진다면 노란색 부분이 옆으로 퍼지며 보다 멀리 떨어진 point에도 더 많은 가중치를 주게 된다.

Ex) Gaussian kernel이라면,

$$k_\lambda(\mathbf{x_1}, \mathbf{x_2}) = exp(-||\mathbf{x_1} - \mathbf{x_2}||^2/2\lambda^2)$$

따라서, 결과적으로 λ가 커질수록 초록색 fitted line이 smooth해지게 된다.

Radial basis Function Networks



Nadaraya-Watson model의 경우 특정한 분포를 가정하고 시작하지 않았기 때문에 대표적인 non-parametric model이다.

하지만, *Nadaraya-Watson* model도 한계가 있는데, 바로 한 data poing를 예측할 때마다 다른 모든 training data와의 거리를 계산해야 하기 때문에, 기본적으로 연산량이 많고 training data가 많아질수록 연산량도 기하급수적으로 늘어나게 된다.
이처럼 연산량이 많다는 것은 non-parametric model들이 공유하는 공통적인 한계점이고 이는 뒤에서 소개할 Gaussian Process또한 마찬가지이다.

4

Gaussian Process

Gaussian Process Regression, Gaussian Process Classification,
Extension to Neural Network

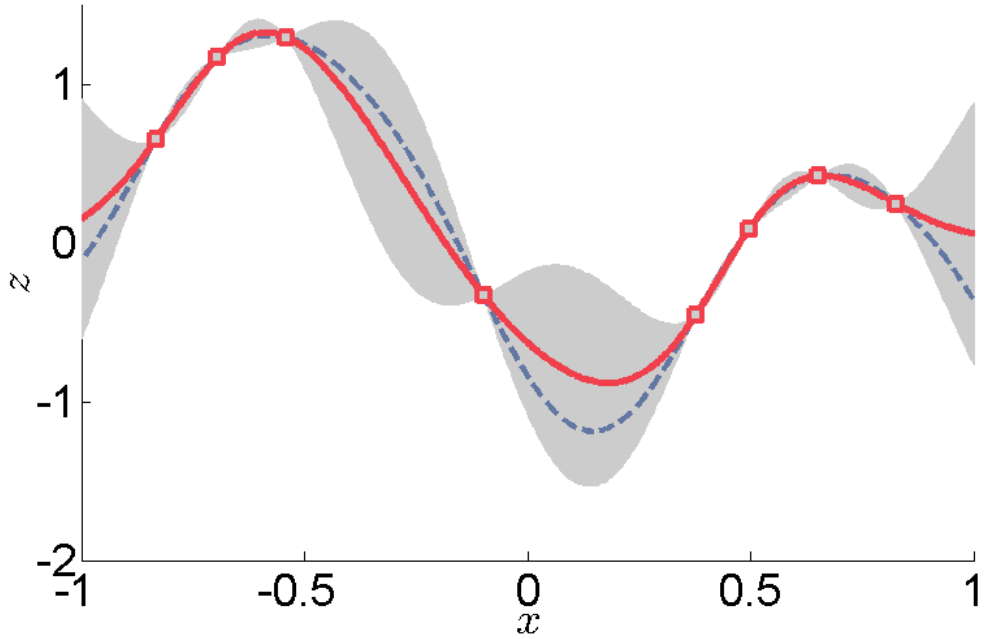
Introduction

- 1절에서는 duality의 개념을 non-probabilistic regression model에 적용함으로써 kernel을 소개하였다. 이 장에서는 probabilistic discriminative model에서의 kernel의 역할에 대해 소개하며, 이를 확장하여 Gaussian Process(GP)에 대한 설명으로 나아갈 것이다. 결과적으로 Bayesian viewpoint에서 kernel이 어떻게 자연스럽게 등장하는지 살펴볼 것이다.
- $y(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \phi(\mathbf{x})$의 형태를 가지는 회귀 모델을 떠올려보자. 이때 $\mathbf{w}$는 매개변수의 벡터이며, $\phi(\mathbf{x})$는 $\mathbf{x}$에 종속적인 고정된 nonlinear basis function이었다. 우리는 $\mathbf{w}$에 대한 prior distribution으로부터 $y(\mathbf{x}, \mathbf{w})$에 대한 prior distribution을 유도하였고, 또한 training set이 주어졌을 때 $\mathbf{w}$에 대한 posterior distribution을 계산하고 regression function에 대한 prior도 구할 수 있었다. 그리고 이를 바탕으로 새로운 입력 벡터 $\mathbf{x}$에 대한 predictive distribution인 $p(t|\mathbf{x})$를 도출할 수 있었다.

(방학 세션 Week3 참고)

Introduction

- GP에서는 매개변수(회귀 모델의 $\mathbf{w}$)를 생략한다. 대신 함수들에 대한 prior를 직접 정의하는데, 함수 공간상에서의 분포를 고려하는 것이 매우 어렵게 느껴진다. 함수 공간이라는 것 자체가 셀 수 없이 많은 함수를 담고 있기 때문이다. 하지만 실제로는 training set과 test set의 data points에 해당하는 $\mathbf{x}_n$의 이산 집합에서의 함숫값에 대해서만 고려하면 되므로, 유한한 공간에서 함수를 다루게 된다.
- 공간통계의 Gaussian Process Regression(GPR) 모델인 kriging (Cressie, 1993), ARMA, Kalman filter, RBF network 등도 전부 GP의 일종이라 할 수 있겠다.
- 그렇다면 Gaussian Process는 정확하게 무엇일까?



Example of one-dimensional data interpolation by kriging, with credible intervals. Squares indicate the location of the data. The kriging interpolation, shown in red, runs along the means of the normally distributed credible intervals shown in gray. The dashed curve shows a spline that is smooth, but departs significantly from the expected values given by those means.

Gaussian Process

- Def. A random process $X(t)$ is a **Gaussian Process** if, $\forall k \in \mathbb{N}$ and $\forall t_1, t_2, \cdots, t_k$, a random vector formed by $X(1), X(2), \cdots X(t_k)$ is jointly Gaussian.
- The joint density is completely specified by

  Mean: $m(t) = E(X(t))$,

  Covariance: $k(t,s) = cov(X(t), X(s))$.
- Notation. $X(t) \sim GP(m(t), k(t,s))$.
- Example. $X(t) = tA$, where $A \sim N(0,1)$ and $t \in \mathbb{R}$. Then

  Mean: $m(t) = E\big(X(t)\big) = E(tA) = tE(A) = 0$,

  Covariance: $cov\big(X(t), X(s)\big) = cov(tA, sA) = ts$.

  X(t) is non-stationary.

(If being stationary, $k()$ must be a function of $||t-s||$.)

- Def. (**Kernel**) Let $\mathcal{X}$ be a non-empty set. A function $k: \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a kernel if there exists a Hilbert space $\mathcal{H}$ and a map $\phi: \mathcal{X} \to \mathcal{H}$ such that $\forall x, x' \in \mathcal{X}$,

$$k(x, x') \cong \langle \phi(x), \phi(x') \rangle_{\mathcal{H}}.$$

Linear Regression Revisited

$y(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x})$의 형태를 가지는 회귀 모델을 생각해보자. $\mathbf{w}$에 대한 prior distribution으로

$$p(\mathbf{w}) = N(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I})$$

을 설정하자. 여기서 α는 precision hyperparameter이다. 우리는 위의 prior를 바탕으로 함수 $y(\mathbf{x})$에 대한 확률 분포를 도출할 수 있다. 실전에서는 이 함수의 값을 특정 $\mathbf{x}$에 대해 계산하는 것이 필요하다. 가령 Training data points $\mathbf{x}_1, \ldots, \mathbf{x}_N$에 대해서, 함숫값 $y(\mathbf{x}_1), \ldots, y(\mathbf{x}_N)$의 joint distribution에 관심을 갖게 될 것이다. 이 분포를 각각의 원소가 $y_n = y(\mathbf{x}_n)$, for $n \in [N]$인 벡터 $\mathbf{y}$로 지칭하자. 그렇다면 $y(\mathbf{x})$는 다음과 같이 주어지게 된다:

$$\mathbf{y} = \mathbf{\Phi}\mathbf{w}.$$

여기서 $\mathbf{\Phi}$는 design matrix이며 $\mathbf{\Phi}_{nk} = \phi_k(\mathbf{x}_n)$이다.

이제 $\mathbf{y}$의 분포를 찾아보자.

$$\mathrm{E}[\mathbf{y}] = \mathbf{\Phi}\mathrm{E}[\mathbf{w}] = \mathbf{0},$$
$$\mathrm{cov}[\mathbf{y}] = \mathrm{E}[\mathbf{y}\mathbf{y}^T] = \mathbf{\Phi}\mathrm{E}[\mathbf{w}\mathbf{w}^T]\mathbf{\Phi}^T =: \mathbf{K}.$$

여기서 $\mathbf{K}$는 그램 행렬이며, 그 원소는 다음과 같다.

$$\mathbf{K}_{nm} = k(\mathbf{x}_n, \mathbf{x}_m) = \frac{1}{\alpha}\phi(\mathbf{x}_n)^T\phi(\mathbf{x}_m) \quad .$$

이 모델은 GP의 한 예시라 할 수 있다. 앞서 살펴 본 GP의 정의와 같이, GP는 함수 $y(\mathbf{x})$에 대한 joint Gaussian distribution으로 정의된다. 만약 벡터 $\mathbf{x}$가 2차원일 경우, 이 모델을 **Gaussian Random Field**라 한다. 더 일반적으로 **Stochastic Process** $y(\mathbf{x})$는 어떤 유한 집합 $\{y(\mathbf{x}_1), \ldots, y(\mathbf{x}_N)\}$에 대해 일관되게 joint distribution을 부여하는 방식으로 정의된다.

Linear Regression Revisited

$$\mathrm{E}[\mathbf{y}] = \boldsymbol{\Phi}\mathrm{E}[\mathbf{w}] = \mathbf{0},$$

$$\mathrm{cov}[\mathbf{y}] = \mathrm{E}[\mathbf{y}\mathbf{y}^T] = \boldsymbol{\Phi}\mathrm{E}[\mathbf{w}\mathbf{w}^T]\boldsymbol{\Phi}^T =: \mathbf{K}.$$

여기서 $\mathbf{K}$는 그램 행렬이며, 그 원소는 다음과 같다.

$$\mathbf{K}_{nm} = k(\mathbf{x}_n, \mathbf{x}_m) = \frac{1}{\alpha}\phi(\mathbf{x}_n)^T\phi(\mathbf{x}_m).$$

이 모델은 GP의 한 예시라 할 수 있다. 앞서 살펴 본 GP의 정의와 같이, GP는 함수 $y(\mathbf{x})$에 대한 joint Gaussian distribution으로 정의된다. 만약 벡터 $\mathbf{x}$가 2차원일 경우, 이 모델을 **Gaussian Random Field**라 한다. 더 일반적으로 **Stochastic Process** $y(\mathbf{x})$는 어떤 유한 집합 $\{y(\mathbf{x}_1), \ldots, y(\mathbf{x}_N)\}$에 대해 일관되게 joint distribution을 부여하는 방식으로 정의된다.

Linear Regression Revisited

GP의 핵심은 N개의 변수 $y_1, \ldots, y_N$의 joint dist.가 평균과 공분산만으로 완벽하게 정의되어야 한다는 것이다. 그런데 실전에서는 $y(\mathbf{x})$의 평균값에 대해 아무런 사전 지식도 없을 것이다. 따라서 그냥 0으로 설정하며,이는 $E[\mathbf{w}|\alpha]$를 0으로 설정하는 것과 동일한 이유이다. 그 후 공분산을 어떤 두 x 값에 대해 계산하게 되는데, 이때 이 계산값은 커널 함수를 통해 주어지게 된다:

$$E[y(\mathbf{x}_n)y(\mathbf{x}_m)] = k(\mathbf{x}_n, \mathbf{x}_m).$$

만약 $p(\mathbf{w}) = N(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I})$라는 prior와 $y(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x})$ 형태의 회귀 모델에 의해 정의된 GP의 경우, 커널 함수는

$$\mathbf{K}_{nm} = k(\mathbf{x}_n, \mathbf{x}_m) = \frac{1}{\alpha}\phi(\mathbf{x}_n)^T\phi(\mathbf{x}_m)$$

의 형태로 생각할 수 있다.

* 다른 커널로는 무엇이 있을까?

이외에도 많이 쓰이는 커널로는 Gaussian kernel과 Exponential kernel이 있으며, 각각 다음과 같다:

Gaussian: $k(\mathbf{x}_n, \mathbf{x}_m) = \exp(-||\mathbf{x}_n - \mathbf{x}_m||^2/2\sigma^2)$

Exponential: $k(\mathbf{x}_n, \mathbf{x}_m) = \exp(-\theta||\mathbf{x}_n - \mathbf{x}_m||)$

Gaussian Processes for Regression (GPR)

회귀 모델에서 random noise variable은 다음과 같이 정의된다:

$$t_n = y_n + \epsilon_n.$$

노이즈가 정규분포를 따름을 고려하면,

$$p(t_n|y_n) = N(t_n|y_n, \beta^{-1})$$

가 되며, β는 노이즈의 precision을 의미하는 hyperparamter이다. 노이즈는 i.i.d.로 가정되므로, N개의 데이터에 대해, $\mathbf{y} = (y_1, \dots y_N)^T$에 조건부인 $\mathbf{t} = (t_1, \dots t_N)^T$의 joint distribution은 다음 형태의 isotropic Gaussian distribution으로 나타낼 수 있다:

$$p(\mathbf{t}|\mathbf{y}) = N(\mathbf{t}|\mathbf{y}, \beta^{-1}\mathbf{I}_N).$$

GP의 정의에 따라서 marginal distribution $p(\mathbf{y})$는 평균이 0이고 공분산이 그램 행렬 $\mathbf{K}$인 Gaussian dist.를 따른다:

$$p(\mathbf{y}) = N(\mathbf{y}|\mathbf{0}, \mathbf{K}).$$

GP를 위해 사용되는 kernel function은 $\mathbf{x}_i$와 $\mathbf{x}_j$가 비슷할수록 $cov(y(\mathbf{x}_i), y(\mathbf{x}_j))$도 높아지도록 선택되어야 한다. 단, '비슷하다'의 정의를 무엇으로 할지는 적용 사례에 따라 달라지게 된다.

* GP vs. GPR

Gaussian Process Regression은 Gaussian Process의 성질을 사용하는 nonparametric Bayesian Regression의 일종으로, Gaussian Process와는 다르다.

Gaussian Processes for Regression (GPR)

Marginal distribution $p(\boldsymbol{t})$는

$$p(\boldsymbol{t}) = \int p(\mathbf{t}|\mathbf{y})p(\mathbf{y})\,d\mathbf{y} = N(\mathbf{y}|\mathbf{0}, \mathbf{C})$$

(2.115)을 바탕으로 구할 수 있으며, 공분산 행렬 $\mathbf{C}$의 원소는 다음과 같다:

$$C(\mathbf{x}_n, \mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m) + \beta^{-1}\delta_{nm}.$$

이 결과는 $y(x)$에 연관된 임의성에 대한 Gaussian 분포와 ϵ에 연관된 임의성에 대한 Gaussian 분포가 서로 독립적이므로, 따라서 각각의 공분산을 단순히 더하기만 하면 된다는 것을 보여준다.

GPR을 위해 많이 쓰이는 커널 함수는

$$k(\mathbf{x}_n, \mathbf{x}_m) = \theta_0 \exp\{-\frac{\theta_1}{2}||\mathbf{x}_n - \mathbf{x}_m||\} + \theta_2 + \theta_3\mathbf{x}_n^T\mathbf{x}_m$$

인데, 이는 이차 형태에 지수 함수를 취하고 여기에 constant term θ_2, linear term $\theta_3\mathbf{x}_n^T\mathbf{x}_m$을 더한 것이다.

*How to get $p(\boldsymbol{t})$ (Formula 2.115)

Given $p(\mathbf{y}) = N(\boldsymbol{\mu}, \boldsymbol{\Lambda}^{-1}) = N(\mathbf{0}, \mathbf{K})$ and

$$p(\mathbf{t}|\mathbf{y}) = N(\mathbf{A}\mathbf{y} + \mathbf{b}, \mathbf{L}^{-1}) = N(\mathbf{t}|\mathbf{y}, \beta^{-1}\mathbf{I}_N),$$

$$p(\boldsymbol{t}) = \int p(\mathbf{t}|\mathbf{y})p(\mathbf{y})\,d\mathbf{y} = N\left(\mathbf{A}\boldsymbol{\mu} + \mathbf{b}, \mathbf{L}^{-1} + \boldsymbol{A}\boldsymbol{\Lambda}^{-1}\boldsymbol{A}^{\boldsymbol{T}}\right)$$

*왜 $\theta_0 \exp\{-\frac{\theta_1}{2}||\mathbf{x}_n - \mathbf{x}_m||\} + \theta_2 + \theta_3\mathbf{x}_n^T\mathbf{x}_m$가 많이 쓰일까?

linear term과 constant term이 모델을 더 유연하게 만들어주며, 데이터가 선형적일 때 모델의 fitting을 높일 수 있다. 또한 input vector 사이의 거리가 지수 함수의 형태를 취하면서, 둘의 거리가 커질수록 커널 값이 빠르게 감소하게 되는데, 이는 이전 장에서 살펴본 kernel의 선택 기준에 적합함.

Gaussian Processes for Regression (GPR)

이제 GPR을 통해 새로운 input variable에 대한 target variable을 예측해보자. 이를 위해서는 GPR의 predictive regression $p(t_{N+1}|\mathbf{t}_N)$을 계산해야 한다. 이를 위해 우선 joint distribution $p(\mathbf{t}_{N+1})$을 구한 뒤, PRML 2.3.1절의 결과를 적용하면 필요한 조건부 분포를 구할 수 있다.

앞선 $p(\boldsymbol{t})$의 사례로부터, 우리는 $p(\boldsymbol{t}_{N+1})$의 joint distribution을 알 수 있고, 이는 다음과 같다:

$$p(\boldsymbol{t}_{N+1}) = N(\mathbf{t}_{N+1}|\mathbf{0}, \mathbf{C}_{N+1}).$$

여기서 $\mathbf{C}_{N+1}$은 $(N+1)\times(N+1)$ 공분산 행렬이며, 원소는 $p(\boldsymbol{t})$에서의 원소와 같다. 공분산 행렬을 다음과 같이 partition하자:

$$\mathbf{C}_{N+1} = \begin{pmatrix} \mathbf{C}_N & \mathbf{k} \\ \mathbf{k}^T & c \end{pmatrix}.$$

벡터 $\mathbf{k} \in \mathbb{R}^{\boldsymbol{n}}$의 i번째 원소는 $k(\mathbf{x}_i, \mathbf{x}_{N+1})$이며, $c \in \mathbb{R}$은 $k(\mathbf{x}_{N+1}, \mathbf{x}_{N+1}) + \beta^{-1}$이다.

Gaussian Processes for Regression (GPR)

이제 PRML (2.81), (2.82)의 결과물을 이용하면 $p(t_{N+1}|\mathbf{t})$가 Gaussian dist.이며 평균과 공분산이 다음과 같음을 알 수 있다:

Mean: $m(\mathbf{x}_{N+1}) = \mathbf{k}^T \mathbf{C}_N^{-1} \mathbf{t}$,

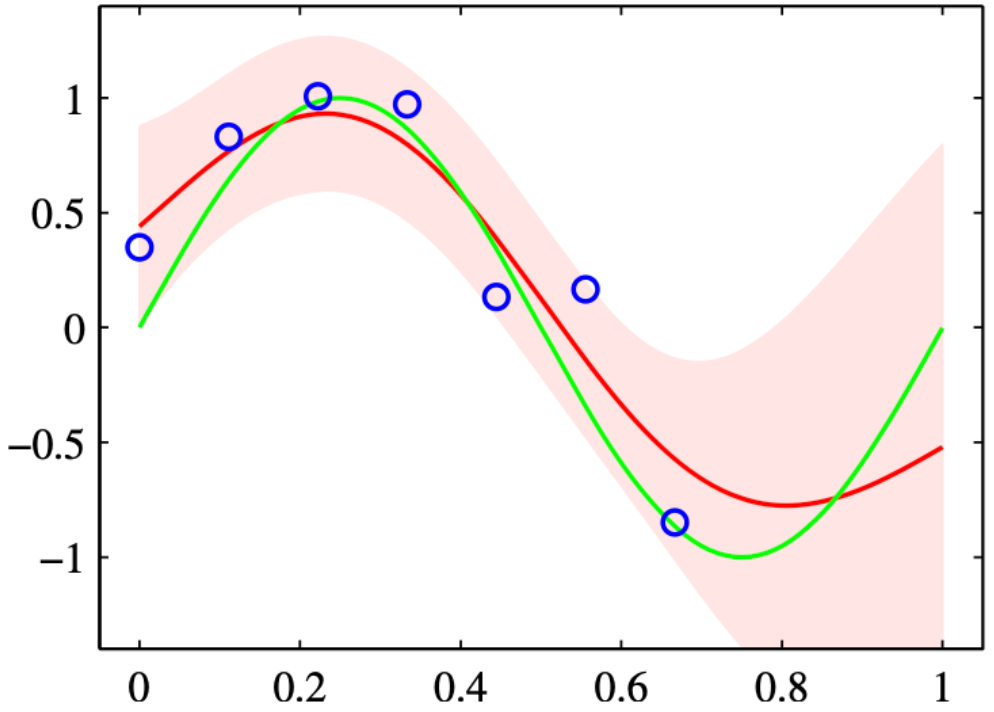
Cov: $\sigma^2(\mathbf{x}_{N+1}) = c - \mathbf{k}^T \mathbf{C}_N^{-1} \mathbf{t}$.

이것이 GPR의 핵심인데, 다시 한번 벡터 $\mathbf{k}$를 살펴보자.

$(\mathbf{k})_i = k(\mathbf{x}_i, \mathbf{x}_{N+1})$이고, 결국 $\mathbf{k}$는 새로운 input $\mathbf{x}_{N+1}$에 대한 함수이다. 그렇다는 것은 predictive distribution의 평균과 분산이 모두 $\mathbf{x}_{N+1}$에 종속적인 Gaussian dist.임을 알 수 있다. 또한 이를 표현하기 위해 위의 $m(\mathbf{x}_{N+1})$를 다음과 같이 표현할 수도 있다:

$$m(\mathbf{x}_{N+1}) = \sum_{i=1}^{N} a_n \, k(\mathbf{x}_i, \mathbf{x}_{N+1}), \qquad a_n = (\mathbf{C}_N^{-1}\mathbf{t})_n.$$

따라서 만약 커널 함수가 $||\mathbf{x}_n - \mathbf{x}_m||$에만 종속적이라고 한다면, 우리는 최종적으로 RBF의 확장형을 얻을 수 있다.



In the illustration of GPR applied to the sinusoidal dataset, the three rightmost data points have been excluded. The blue dots represent the sampled data points obtained by adding Gaussian noise to a sinusoidal function (shown in green). The red line depicts the mean of the Gaussian process predictive distribution, while the shaded region indicates the area within plus and minus two standard deviations. Uncertainty increases noticeably to the right of the observed data points.

Gaussian Processes for Regression (GPR)

GP에 있어 가장 중요한 것은 $N \times N$ 크기 행렬의 역을 계산하는 것이다. 이는 표준 방법을 사용할 경우 $O(N^3)$을 요구한다. 반면 basis function model에서 계산해야 하는 S_N^{-1}은 $M \times M$ 크기 행렬이므로 $O(M^3)$의 복잡도를 가진다. 두 관점 모두 역행렬 계산은 하나의 훈련집합에 대해 한 번 실행되어야 한다. 두 방법 모두 벡터-행렬 곱이 필요하고, 이는 각각 $O(N^2)$, $O(M^3)$ 의 코스트를 가진다. 만약 $M < N$이라면 basis function framework를 사용하는 것이 효율적일 것이다.

하지만 GP의 장점 중 하나는 무한 개의 기저 함수로만 표현할 수 있는 공분산 함수 (kernel fct)를 고려할 수 있다는 것이다. 즉, 유한 개의 기저 함수라는 제약이 없으므로 GPR은 어떤 문제에든 적용될 수 있으며, 도메인으로 시간이나 공간을 가지더라도 표현할 수 있다.

Learning the Hyperparameters

GP 모델의 예측력은 kernel의 선택에 의존한다. 하지만 실제로는 kernel로 고정된 함수를 사용하는 대신, 매개변수적 함수를 사용하고, 데이터로부터 매개변수를 추정하는 것이 더 좋을 수도 있다. 이러한 매개변수는 상관 관계의 길이 스케일과 노이즈의 정밀도와 같은 것들을 조절하며, 표준 매개변수 모델에서의 하이퍼파라미터에 해당한다.

하이퍼파라미터 튜닝의 기본적인 아이디어는 likelihood fct. $p(\mathbf{t}|\boldsymbol{\theta})$를 바탕으로 mle를 찾는 것이다. 이는 conjugate gradients(Fletcher, 1987; Nocedal and Wright, 1999; Bishop and Nabney, 2008)와 같은 그레디언트 기반 최적화 알고리즘을 통해 수행할 수 있다.

Learning the Hyperparameters

GPR 모델의 log-likelihood는 다변량 정규분포를 기반으로 쉽게 찾을 수 있다:

$$\ln p(\mathbf{t}|\boldsymbol{\theta}) = -\frac{1}{2}\ln|\mathbf{C}_N| - \frac{1}{2}\mathbf{t}^T\mathbf{C}_N^{-1}\mathbf{t} - \frac{N}{2}\ln(2\pi).$$

이제 최적화를 위해 param. vector $\boldsymbol{\theta}$에 대한 그레디언트를 찾아보자:

$$\frac{\partial}{\partial\theta_i}\ln p(\mathbf{t}|\boldsymbol{\theta}) = -\frac{1}{2}\mathrm{tr}\left(\mathbf{C}_N^{-1}\frac{\partial\mathbf{C}_N}{\partial\theta_i}\right) + \frac{1}{2}\mathbf{t}^T\mathbf{C}_N^{-1}\frac{\partial\mathbf{C}_N}{\partial\theta_i}\mathbf{C}_N^{-1}\mathbf{t}.$$

미분 과정은 (C.21)과 (C.22) 참조. 일반적으로 $\ln p(\mathbf{t}|\boldsymbol{\theta})$는 non-convex fct이기 때문에 최댓값이 여러개일 수 있다.

$\boldsymbol{\theta}$에 대한 prior를 도입하고 gradient-based method로 log-posterior를 최대화할 수도 있다. Bayesian 관점에서는 $p(\boldsymbol{\theta})$와 $p(\mathbf{t}|\boldsymbol{\theta})$를 곱해 marginalize하여 posterior를 구하고, 여기에 log를 취해 최대화할 수 있다. 그러나 정확한 marginalization은 불가능하므로 적당한 근사치를 구하는 데서 만족해야 한다.

*How to differentiate a matrix (Appendix C)

$$\frac{\partial}{\partial x}(\mathbf{A}^{-1}) = -\mathbf{A}^{-1}\frac{\partial\mathbf{A}}{\partial x}\mathbf{A}^{-1}$$

$$\frac{\partial}{\partial x}\ln|\mathbf{A}| = \mathrm{tr}\left(\mathbf{A}^{-1}\frac{\partial\mathbf{A}}{\partial x}\right)$$

Learning the Hyperparameters

GPR은 평균과 분산이 input vector $\mathbf{x}$의 함수로 표현되는 예측 분포를 도출하게 된다. 하지만 한 가지 다시 고려해야 할 점은 이 도출 과정에서 우리가 additive noise로부터 발생된 predictive variance가 상수 파라미터 β에 의해 조절된다고 가정했다는 것이다. **Heteroscedastic**(이분산성)이라 알려져 있는 몇몇 케이스에서는 variance가 상수가 아니라 $\mathbf{x}$에 대한 함수일 수도 있고, 이 경우 우리는 β를 설명하기 위한 새로운 GP를 도입할 필요가 있다 (Goldbert et al., 1998). GP를 이용하는 이유는 β가 분산이어서 non-negative이기 때문에, GP를 이용하여 $\ln \beta(\mathbf{x})$를 모델링하게 된다.

Gaussian Processes for Classification (GPC)

확률적 분류 모델의 목표는 새 input vector에 대한 target variable의 posterior probability를 모델링하는 것이다. 이 확률들은 $(0,1)$ 구간에 존재야 한다. 하지만 GP의 output range는 실수 전체이므로, 적절한 non-linear activation fct를 적용함으로써 분류 문제에 GP를 적용할 수 있다.

우리의 목표는 predictive distribution $p(t_{N+1}|\mathbf{t})$를 찾는 것이다. 우선 벡터 $\mathbf{a}_{N+1} = \big(a(x_1), \dots, a(x_{N+1})\big)$의 prior에 대한 GP를 도입해보자. 이는 $\mathbf{t}_{N+1}$에 대한 non-GP를 정의해주며, 이를 훈련 데이터 $\mathbf{t}_N$에 대한 조건부로 만들면 predictive dist.를 구할 수 있게 된다. $\mathbf{a}_{N+1}$에 대한 GP prior는 다음과 같다:

$$p(\mathbf{a}_{N+1}) = N(\mathbf{a}_{N+1}|\mathbf{0}, \mathbf{C}_{N+1}).$$

Gaussian Processes for Classification (GPC)

GPR과는 달리 Cov. matrix가 더 이상 noise term을 포함하지 않는다. 왜냐하면 Classification problem에서는 모든 training data points가 올바른 label을 가지고 있다고 가정하기 때문이다. 하지만 수치적 이유로 noise-like term을 도입하는 것이 때에 따라 편리할 수 있다. 따라서 공분산 행렬 $\mathbf{C}_{N+1}$는 다음 원소를 가지게 된다:

$$C(\mathbf{x}_n, \mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m) + \nu\delta_{nm}.$$

보통 ν의 값은 미리 고정되며, 커널 함수 $k(\cdot,\cdot)$는 param. vector $\boldsymbol{\theta}$에 의해 조절된다고 가정한다. Training set으로부터 $\boldsymbol{\theta}$를 학습하는 방법은 이후에 논의될 것이다.

Gaussian Processes for Classification (GPC)

Binary Classification의 경우, $p(t_{N+1}=1|\mathbf{t}_N)$만 예측하면 되는데, $p(t_{N+1}=0|\mathbf{t}_N)=1-p(t_{N+1}=1|\mathbf{t}_N)$ 이기 때문이다. 이 경우 predictive dist.는 다음과 같다:

$$p(t_{N+1}=1|\mathbf{t}_N)$$

$$=\int p(t_{N+1}=1|\mathrm{a}_{N+1})\,p(a_{N+1}=1|\mathbf{t}_N)\,d\mathbf{a}_{N+1}. \qquad (6.76)$$

일반적으로 확률 문제를 위해 logistic fct.을 이용하므로, 위에서 $p(t_{N+1}=1|\mathrm{a}_{N+1})=\sigma(\mathrm{a}_{N+1})$이다.

위 적분은 직접 계산해내기 어렵고, 따라서 근사를 이용해야 한다. 이때 고려할 수 있는 방법들에는 다음 것들이 있다:

- Sampling을 통한 근사 (Neal. 1997)
- Analytical Approxmiation에 기반한 근사 (Section 4.5.2)

Gaussian Processes for Classification (GPC)

다만 우리는 posterior $p(a_{N+1}|\mathbf{t}_N)$에 대한 Gaussian Approximation이 있어야 한다. Posterior를 Gaussian으로 근사화할 수 있는 근거는 데이터 포인트 수가 증가함에 따라 CLT의 결과로 true posterior가 Gaussian으로 수렴하기 때문이다.

하지만 GP의 경우, 변수의 수가 데이터 포인트 수와 함께 증가하므로 이러한 주장을 직접 적용하기는 어렵다. 그러나 데이터 포인트 수 증가는 곧 $\mathbf{x}$ space의 어떤 fixed region에 속하는 포인트의 숫자가 늘어나는 것임을 고려하면, 이에 해당하는 함수 $a(\mathbf{x})$의 불확실성은 감소하게 될 것이다. 이를 바탕으로 asymptotically하게 Gaussian dist.로 수렴한다고 할 수 있다 (Williams and Barber, 1998).

Gaussian Approximation을 위하여 세 가지 다른 접근법이 고려되었다:

- Variational Inference (Gibbs and MacKay, 2000)
- Expectation Propagation (Opper an Winther, 2000b; Minka, 2001b; Seeger, 2003)
- Laplace Approximation

Gaussian Processes for Classification (GPC)

- Variational Inference(변분 추론법)을 이용해서 시그모이드 함수의 곱을 가우시안의 곱으로 근사할 수 있다. 따라서 $\mathbf{a}_N$에 대한 marginalization이 해석적으로 가능하게 되고, 또한 이를 통해 likelihood fct $p(\mathbf{t}_N|\boldsymbol{\theta})$의 infimum을 알 수도 있다. Softmax fct에 대해 Gaussian Approximation을 적용함으로써 GPC에의 변분 추론법을 다중 클래스($K > 2$) 문제에 대해 확장할 수도 있다(Gibbs, 1997).
- Expectation Propagation(기대 전파법)을 이용하여 근사화할 수도 있다. EP는 복잡한 joint dist.를 여러 개의 단순한 분포로 근사화하는 것으로서, true posterior가 unimodal이기 때문에 EP를 사용하는 방법은 꽤 좋은 결과를 낸다.

Laplace Approximation

Predict dist.를 평가하기 위해서 우리는 Bayes thm.을 이용해 $\mathbf{a}_{N+1}$에 대한 posterior의 Gaussian approx.를 찾을 수 있다:

$$p(a_{N+1}|\mathbf{t}_N) = \int p(a_{N+1}, \mathbf{a}_N|\mathbf{t}_N)\, d\mathbf{a}_N$$

$$= \frac{1}{p(\mathbf{t}_N)} \int p(a_{N+1}, \mathbf{a}_N) p(\mathbf{t}_N|a_{N+1}, \mathbf{a}_N)\, d\mathbf{a}_N$$

$$= \frac{1}{p(\mathbf{t}_N)} \int p(a_{N+1}|\mathbf{a}_N) p(\mathbf{a}_N) p(\mathbf{t}_N|\mathbf{a}_N)\, d\mathbf{a}_N$$

$$= \int p(a_{N+1}|\mathbf{a}_N) p(\mathbf{a}_N|\mathbf{t}_N)\, d\mathbf{a}_N. \qquad (6.77)$$

Laplace Approximation

조건부 분포 $p(a_{N+1}|\mathbf{a}_N)$는 GPR에서의 식으로부터 얻어지며, 다음과 같다:

$$p(a_{N+1}|\mathbf{a}_N) = N(a_{N+1}|\mathbf{k}^T\mathbf{C}_N^{-1}\mathbf{a}_N, c - \mathbf{k}^T\mathbf{C}_N^{-1}\mathbf{k}). \qquad (6.78)$$

따라서 우리는 posterior $p(\mathbf{a}_N|\mathbf{t}_N)$에 대한 라플라스 근사를 찾고 두 정규분포의 convolution에 대한 결과를 사용하여 $p(a_{N+1}|\mathbf{t}_N)$을 구할 수 있다.

사전 분포 $p(\mathbf{a}_N)$는 평균이 0이고 공분산이 $\boldsymbol{C}_N$인 GP로 주어진다. 이때 데이터항은 다음과 같이 주어진다(데이터 간 독립성 가정):

$$p(\mathbf{t}_N|\mathbf{a}_N) = \prod_{n=1}^{N} \sigma(a_n)^{t_n}(1-\sigma(a_n))^{1-t_n} = \prod_{n=1}^{N} e^{a_n t_n}\sigma(-a_n)\,.$$

Laplace Approximation

$p(\mathbf{a}_N|\mathbf{t}_N)$의 로그에 Taylor expansion을 적용함으로써 라플라스 근사치를 구할 수 있다. 이때 additive normalization constant까지의 결과는 다음과 같다:

$$\boldsymbol{\Psi}(\mathbf{a}_N) = \ln p(\mathbf{a}_N) + \ln p(\mathbf{t}_N|\mathbf{a}_N)$$

$$= -\frac{1}{2}\mathbf{a}_N^T\mathbf{C}_N^{-1}\mathbf{a}_N - \frac{N}{2}\ln(2\pi) - \frac{1}{2}\ln|\boldsymbol{C}_N| + \mathbf{t}_N^T\mathbf{a}_N - \sum_{n=1}^{N}\ln(1+e^{a_n})$$

$$+ const. \quad (6.80)$$

이후 첫 번째로, mode(posterior)를 찾아야 하는데, 이를 위해서 $\boldsymbol{\Psi}(\mathbf{a}_N)$의 Gradient를 구해야 한다. 이는 다음과 같다:

$$\nabla\boldsymbol{\Psi}(\mathbf{a}_N) = \mathbf{t}_N - \boldsymbol{\sigma}_N - \mathbf{C}_N^{-1}\mathbf{a}_N.$$

$\boldsymbol{\sigma}_N$는 $\sigma(a_n)$을 원소로 가지는 벡터이다. 단순히 기울기를 0으로 설정하는 방식으로는 mode를 찾을 수 없다. 왜냐하면 $\boldsymbol{\sigma}_N$이 $\mathbf{a}_N$에 대해 nonlinearly dependent하기 때문이다. 따라서 Newton-Raphson method와 같은 반복적인 방법에 의존해야 한다.

Laplace Approximation

이때 Iterative Reweighted Least Squares(IRLS) 알고리즘을 사용하게 된다. 이 알고리즘은 이차 미분을 필요로 하는데, 어차피 라플라스 근사를 위해서 필요한 값이다.

$$\nabla^2\Psi(\mathbf{a}_N) = -\mathbf{W}_N - \mathbf{C}_N^{-1}.$$

$\mathbf{W}_N$은 $(\mathbf{W}_N)_{ii} = \sigma(a_n)(1-\sigma(a_n))$이고 나머지는 0인 대각행렬이다. 이를 위해 PRML (4.88)에서 구하였던 로지스틱 시그모이드 함수의 미분 결과를 이용하였으며, 대각 성분들은 $(0, ¼)$ 범위에 속하여 Positive definite이다. 또한 $\mathbf{C}_N$과 그 역행렬 모두 positive definite이므로 Hessian matrix $-\nabla^2\Psi(\mathbf{a}_N)$ 역시 positive definite하다.

이에 따라 posterior $p(\mathbf{a}_N|\mathbf{t}_N)$은 log-convex이자 global maximum에 해당하는 single mode를 가지고 있음을 알 수 있다. 하지만 posterior는 Gaussian은 아닌데, 이는 Hessian이 $\mathbf{a}_N$의 함수이기 때문이다.

Laplace Approximation

이제 (4.92)의 Newton-Raphson 공식을 이용하여 $\mathbf{a}_N$에 대한 iterative update 공식을 구할 수 있고, 이는 다음과 같다:

$$\mathbf{a}_N^{\text{new}} = \boldsymbol{C}_N(\mathbf{I} + \boldsymbol{W}_N\mathbf{C}_N)^{-1}\{\mathbf{t}_N - \boldsymbol{\sigma}_N + \mathbf{W}_N\mathbf{a}_N\}.$$

이 iteration은 mode인 $\mathbf{a}_N^*$에 수렴할 때까지 반복되게 되며, mode에서는 기울기 $\nabla\boldsymbol{\Psi}(\mathbf{a}_N)$가 0이 될 것이다. 따라서 $\mathbf{a}_N^*$은 다음을 만족한다:

$$\mathbf{a}_N^* = \boldsymbol{C}_N(\mathbf{t}_N - \boldsymbol{\sigma}_N).$$

Posterior의 mode $\mathbf{a}_N^*$를 찾고 나면, 다음으로 주어지는 헤시안 행렬을 찾을 수 있다:

$$\mathbf{H} := -\nabla^2\boldsymbol{\Psi}(\mathbf{a}_N) = \mathbf{W}_N + \mathbf{C}_N^{-1}.$$

이때 $\mathbf{W}_N$의 원소는 $\mathbf{a}_N^*$를 이용하여 계산할 수 있다. 이를 바탕으로 posterior $p(\mathbf{a}_N|\mathbf{t}_N)$에 대한 Gaussian approx.를 다음과 같이 정의할 수 있다:

$$q(\mathbf{a}_N) = N(\mathbf{a}_N|\mathbf{a}_N^*, \mathbf{H}^{-1}).$$

Laplace Approximation

이제 이를 식 (6.78)과 결합할 수 있으며, 이를 통해 (6.77)의 적분을 계산할 수 있다. 이는 선형 Gaussian 모델에 해당하므로 (2.115)를 이용하여 다음을 구할 수 있다:

$$\mathrm{E}[a_{N+1}|\mathbf{t}_N] = \mathbf{k}^T(\mathbf{t}_N - \boldsymbol{\sigma}_N),$$
$$\mathrm{var}[a_{N+1}|\mathbf{t}_N] = c - \mathbf{k}^T(\boldsymbol{W}_N^{-1} - \mathbf{C}_N)^{-1}\mathbf{k}$$

이제 $p(a_{N+1}|\mathbf{t}_N)$에 대한 Gaussian dist.를 구하였으니 (4.153)의 결과를 이용해 식 (6.76)의 적분을 근사할 수 있다. 섹션 4.5의 Bayesian Logistic Regression Model에서처럼, 만약 $p(t_{N+1}|\mathbf{t}_N) = 0.5$에 해당하는 decision boundary에만 관심이 있다면, 평균만 고려하고 분산의 효과는 무시할 수 있다.

Laplace Approximation

공분산 함수의 $\boldsymbol{\theta}$를 구하기 위하여 우리는 $p(\mathbf{t}_N|\boldsymbol{\theta})$를 최대화하는 mle를 찾을 수 있다. 이때 log-likelihood와 gradient에 대한 식이 필요할 것이다. 우선 likelihood fct는 다음과 같이 정의된다:

$$p(\mathbf{t}_N|\boldsymbol{\theta}) = \int p(\mathbf{t}_N|\mathbf{a}_N)p(\mathbf{a}_N|\boldsymbol{\theta})\, d\mathbf{a}_N$$

이 적분은 해석적으로 계산하기 어렵고, 따라서 다시 한 번 라플라스 근사법을 활용해야 한다. 식 (4.135)의 결과를 사용하여, log-likelihood 함수의 근사치를 다음과 같이 얻을 수 있다:

$$\ln p(\mathbf{t}_N|\boldsymbol{\theta}) = \boldsymbol{\Psi}(\mathbf{a}_N^*) - \frac{1}{2}\ln\left|\mathbf{W}_N + \mathbf{C}_N^{-1}\right| + \frac{N}{2}\ln(2\pi) \quad (6.90)$$

여기서 $\boldsymbol{\Psi}(\mathbf{a}_N^*) = \ln p(\mathbf{a}_N^*|\boldsymbol{\theta}) + \ln p(\mathbf{t}_N|\mathbf{a}_N^*)$이다. 또한 여기서 $\ln p(\mathbf{t}_N|\boldsymbol{\theta})$의 매개변수 벡터 $\boldsymbol{\theta}$에 대한 gradient를 계산해야 한다. $\boldsymbol{\theta}$가 변하면 $\mathbf{a}_N^*$ 역시 변할 것이므로 이에 따라 기울기에 추가적인 항이 포함될 것이다.

Laplace Approximation

따라서 우리가 식 (6.90)을 $\boldsymbol{\theta}$에 대해 (6.80), (C.21), (C.22)의 결과를 바탕으로 미분하게 되면 다음을 얻는다:

$$\frac{\partial \ln p(\mathbf{t}_N|\boldsymbol{\theta})}{\partial \theta_j}$$

$$= \frac{1}{2}\mathbf{a}_N^{*T}\boldsymbol{C}_N^{-1}\frac{\partial \boldsymbol{C}_N}{\partial \theta_j}\boldsymbol{C}_N^{-1}\boldsymbol{a}_N^* - \frac{1}{2}\mathrm{tr}\left[(\mathbf{I}+\mathbf{C}_N\mathbf{W}_N)^{-1}\mathbf{W}_N\frac{\partial \boldsymbol{C}_N}{\partial \theta_j}\right] \quad (6.91)$$

$\boldsymbol{a}_N^*$의 $\boldsymbol{\theta}$에 대한 종속성에서 발생하는 항을 계산해보자. 라플라스 근사치는 $\boldsymbol{\Psi}(\mathbf{a}_N)$이 $\mathbf{a}_N = \mathbf{a}_N^*$에서 기울기 0을 가지도록 되어 있다. 그렇기 때문에 $\boldsymbol{\Psi}(\mathbf{a}_N^*)$은 기울기에 전혀 기여하지 않게 된다. $\boldsymbol{\Psi}(\mathbf{a}_N^*)$은 $\mathbf{a}_N^*$에 종속적이기 때문이다. 이를 바탕으로 하면 $\boldsymbol{\theta}$의 성분 θ_j에 대한 다음의 기여만이 미분에 남게 된다.

$$-\frac{1}{2}\sum_{n=1}^{N}\frac{\partial \ln\left|\mathbf{W}_N+\mathbf{C}_N^{-1}\right|}{\partial a_n^*}\frac{\partial a_n^*}{\partial \theta_j}$$

$$= -\frac{1}{2}\sum_{n=1}^{N}[(\mathbf{I}+\mathbf{C}_N\mathbf{W}_N)^{-1}\mathbf{C}_N]_{nn}\,\sigma_n^*(1-\sigma_n^*)(1-2\sigma_n^*)\frac{\partial a_n^*}{\partial \theta_j} \quad (6.92)$$

Laplace Approximation

여기서 $\sigma_n^* = \sigma(a_n^*)$이다. 그리고 다시 한 번 식 (C.22)의 결과와 $\mathbf{W}_N$의 정의를 사용하였다. $\mathbf{a}_N^*$의 θ_j에 대한 미분값은 (6.84)의 관계식을 θ_j에 대해 미분함으로써 구할 수 있다.

$$\frac{\partial a_n^*}{\partial \theta_j} = \frac{\partial \mathbf{C}_N}{\partial \theta_j}(\mathbf{t}_N - \boldsymbol{\sigma}_N) - \mathbf{C}_N \boldsymbol{W}_N \frac{\partial a_n^*}{\partial \theta_j} \qquad (6.93)$$

이를 다시 정리하면 다음을 얻을 수 있다.

$$\frac{\partial a_n^*}{\partial \theta_j} = (\mathbf{I} + \mathbf{W}_N \mathbf{C}_N)^{-1} \frac{\partial \mathbf{C}_N}{\partial \theta_j}(\mathbf{t}_N - \boldsymbol{\sigma}_N) \qquad (6.94)$$

식 (6.91), (6.92), (6.94) 조합하면 log-likelihood fct의 기울기를 계산할 수 있으며, 곧 최적의 $\boldsymbol{\theta}$를 구할 수 있다.

Connection to Neural Networks

신경망이 나타낼 수 있는 함수의 범위는 $|\text{hidden units}| = M$에 의해 결정되며, 충분히 큰 M에 대해서는, two-layer network로도 임의의 정확도로 주어진 함수를 근사할 수 있다. 하지만 maximum likelihood의 관점에서, 오버피팅을 피하기 위해 M의 수를 제한해야 한다. 그러나 베이지안 관점에서는 training set의 크기에 따라 네트워크의 매개변수 수를 제한하는 것이 별로 의미가 없다.

BNN에서 매개변수 벡터 $\mathbf{w}$에 대한 prior와 네트워크 함수 $f(x, w)$는 신경망의 output vector $y(x)$에 대한 prior를 생성한다. Neal(1996)은 $\mathbf{w}$에 대한 넓은 범위의 사전분포에 대해, 신경망에 의해 생성된 함수의 분포가 $M \to \infty$에 따라 asymptotically하게 GP로 수렴함을 보였다. 그러나 이 limiting distribution에서 신경망의 output variable들이 독립적이라는 것에 유의해야 한다. 신경망의 장점 중 하나는 output들이 hidden unit을 공유하므로 각 hidden unit의 가중치는 모든 output variable에 영향을 준다. 그러나 이러한 특성은 GP로 수렴하면서 사라지게 된다.

감사합니다