Chapter 12. Gaussian Process

Moon Il-chul
icmoon@kaist.ac.kr

Na Yeong-yeon
duddus15@kaist.ac.kr

Kim Hyemi
khm0308@kaist.ac.kr

2018년 11월 26일

차 례

1 Simple Continuous Domain Analysis

1.1 Continuous Domain Data

현실 세계의 많은 데이터들은 연속된 도메인을 가진다. 주가 지수, 시간, 공간, 시공간 등이 대표적인 예이다. 그러나 관측된 데이터는 불연속적인데, 이러한 경우는 관측된 데이터를 바탕으로 잠재 함수를 추정하여 분석할 수 있다.

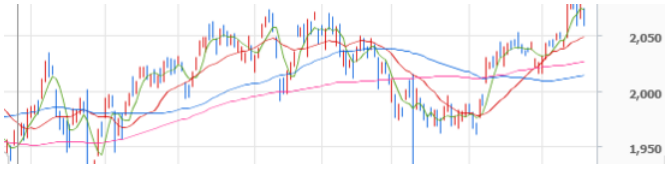


그림 12-1: 연속된 시간에 따른 주가지수

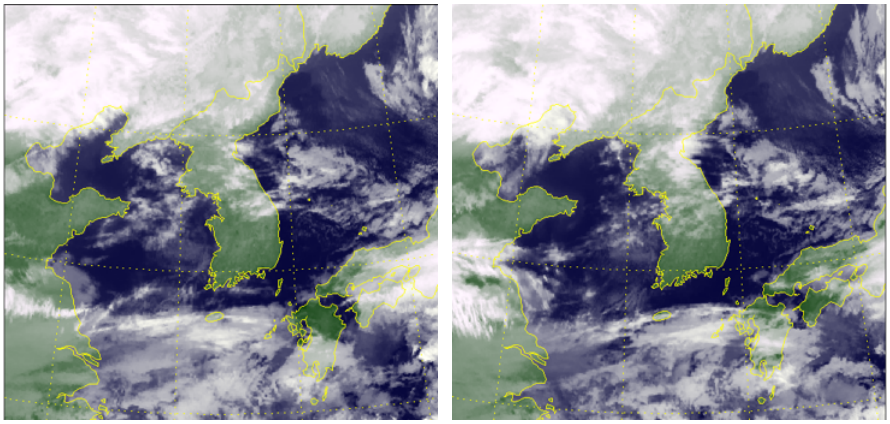


그림 12-2: 연속된 시공간에 따른 구름의 양

1.2 Underlying Function and Observations

잠재 함수를 추정하는데 사용되는 함수로 싱크함수가 있다. 싱크함수의 정의와 그래프는 다음과 같다.

$$sinc(x) = \frac{sin(\pi x)}{\pi x} \tag{12.1}$$

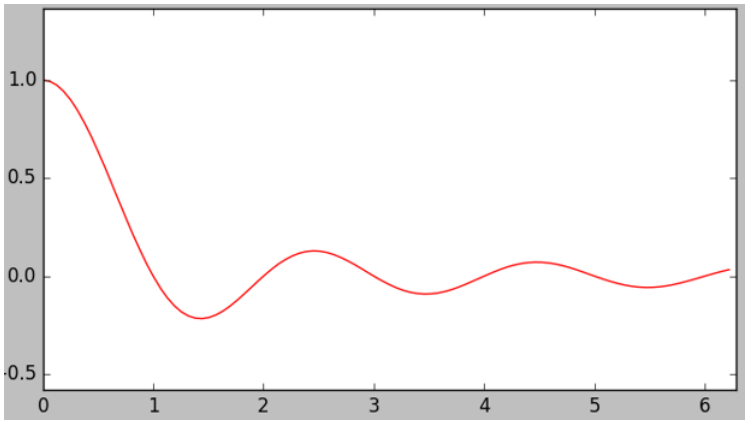


그림 12-3: 싱크함수의 그래프

현실에서는 잠재 함수에 노이즈가 섞여 있다. 따라서 싱크함수에 가우시안 노이즈를 더한 형태를 잠재함수를 추정하는데 사용한다. 도메인의 한 점 x_i에 대한 가우시안 노이즈를 e_i 라고 하면 데이터셋의 확률 분포는 다음과 같다.

$$e_i \sim N(o, \sigma_i^2) \tag{12.2}$$

이를 바탕으로 다음과 같은 두 가지 형태의 관찰데이터셋을 구성해 보겠다.

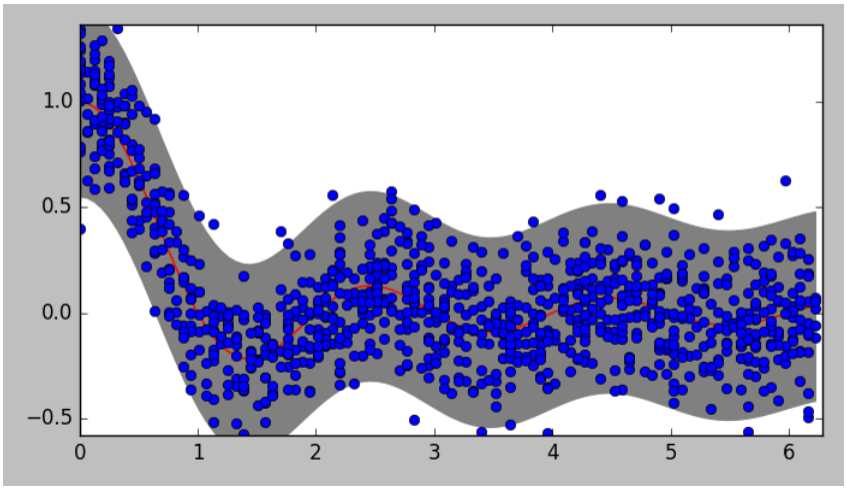


그림 12-4: 싱크함수의 관찰데이터 1

두 데이터셋은 모두 싱크함수를 이용했음에도 불구하고 형태가 다르다. 그림 4의 데이터셋은 모든 x_i에 대해서 $\sigma_i = \sigma$로 일정한 분산 값을 가지는 반면 그림 5의 데이터셋은 x_i 가 증가함에 따라 σ_i도 증가한다. 이로부터 데이터 포인트에

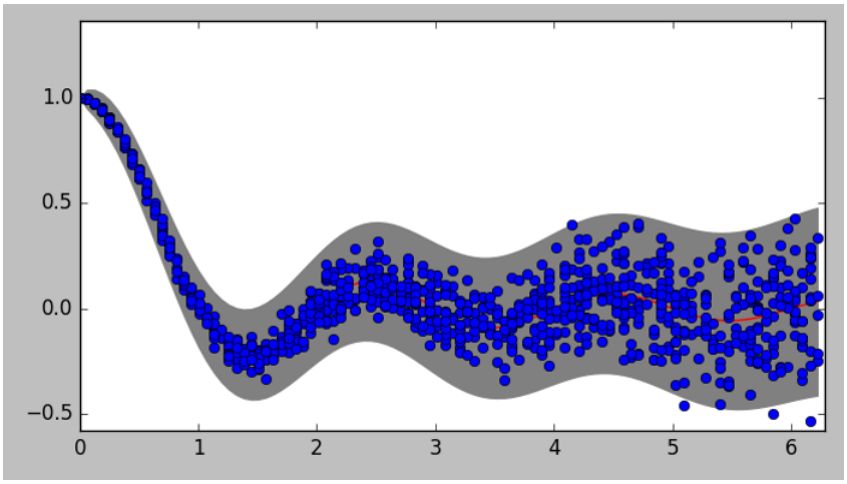


그림 12-5: 싱크함수의 관찰데이터 2

따라 평균과 분산이 단순한 상수가 아니라 함수의 형태로 표현될 수 있음을 알 수 있다. 따라서 관측데이터를 통해 실제 잠재 함수를 추정하기 위해서는 에러항을 모델링해야 한다.

1.3 Simple Analyses without Domain Correlation

1.2 에서는 1000개의 관찰 데이터 포인트를 가지는 데이터셋이었다. 이번 단원에서는 50개의 관찰 데이터 포인트를 가지는 데이터셋을 그림 6과 같이 구성하였다.

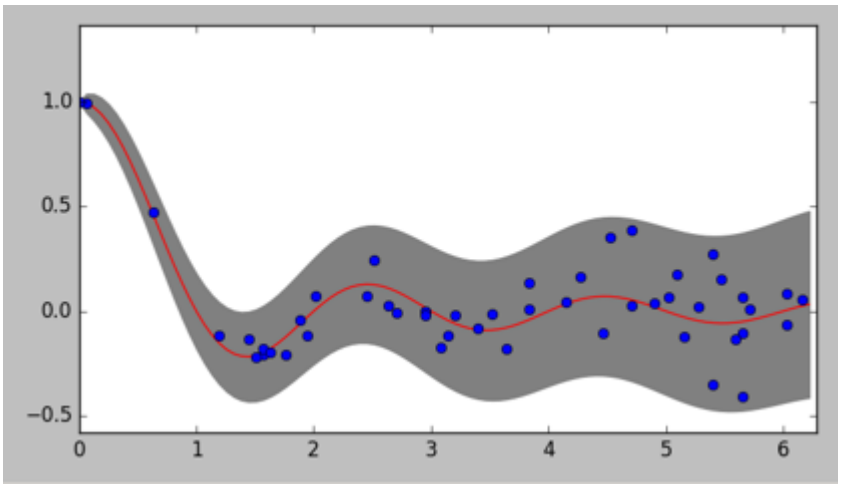


그림 12-6: 싱크함수의 관찰데이터 3

이제 그림 5와 그림 6의 데이터를 바탕으로 잠재함수를 추정해보도록 하겠다.

연속 도메인 중 하나의 포인트 x_i에 대한 관측값이 t_{i1},t_{i2},$\cdots$,t_{ik} 라고 할 때 실제값 y_i는 다음과 같이 추정할 수 있다.

$$y_i = \frac{1}{N}\sum_{j=1}^{k} t_{ij} \tag{12.3}$$

$$N = |\{t_{i1}, t_{i2}, \cdots, t_{ik}\}| \tag{12.4}$$

이처럼 하나의 도메인 포인트에서의 관찰값들의 평균을 내는 방식으로 실제값을 추정할 수 있고, 이렇게 추정한 잠재함수의 그래프는 다음과 같다.

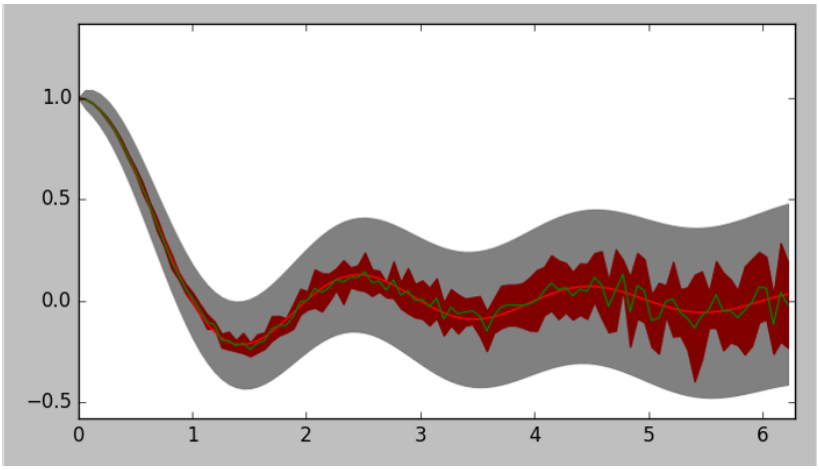


그림 12-7: 관찰데이터2 (그림 5)의 추정된 잠재함수

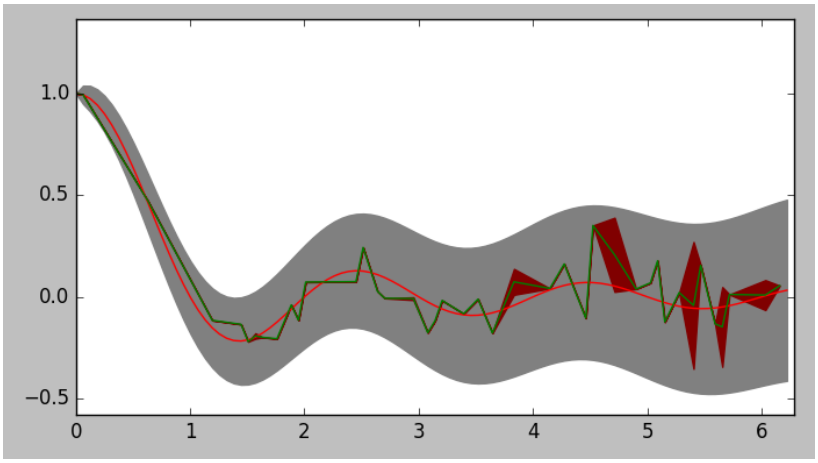


그림 12-8: 관찰데이터3 (그림 6)의 추정된 잠재함수

그림 7의 추정된 함수는 실제 함수가 비슷해 보인다. 이는 이용한 추정 방식이 적합함을 의미한다. 그러나 이를 추정하는데 사용된 그림 5와 같은 관측데이터는 현실에서 접하기 어렵다. 왜냐하면 현실 세계에서는 일반적으로 하나의 도메인 포인트에 대해 한 개 이하의 관찰값만 존재하기 때문이다. 예를 들어 한 개의 센서가 시간별 관찰값을 얻는다고 가정할 때, 특정 시간에 센서의 여러 관찰값을 얻는 것은 불가능하다.
그렇다면 그림 8을 살펴보자. 이 데이터셋에서는 하나의 도메인 포인트에 하나의 관찰값 밖에 없기 때문에 실제값 y_i는 다음과 같은 값을 가진다.

$$y_i = t_i \tag{12.5}$$

그러나 이는 관찰값이 실제값과 동일할 것이라는 단순한 추정이며 관측값들을 직선으로 이어놓은 것에 불과하다.이 추정 방법의 문제점은 도메인 정보를 전혀 사용하지 않은 것이다. 즉, 과거의 관찰값이 현재의 관찰값에 영향을 미칠 수 있음에도 불구하고 사용하지 않았다.

1.4 Simple analyses with Domain Correlation

이제부터는 추정에 현실세계에서 있을법한 데이터인 그림 6의 데이터셋만을 활용하겠다. 여기에서는, 과거의 관찰값이 현재의 관찰값에 영향을 미칠 수 있다는 사실을 감안하여, 도메인 사이의 관계를 고려한 이동 평균(Moving Average)을 활용한다.
이동 평균이란, 시계열의 각 지점에 대하여 그것을 중심으로 하는 전후 일정 지점의 평균값을 구하여 경향성을 도출하는 방법이다. 연속 도메인 위의 한 점 x에 대해 x의 앞 뒤로 도메인을 어느정도 고려할지는 time window $[w_{low}, w_{high}]$를 설정하여 정한다. 연속 도메인 위의 한 점 x에 대해 time window $[w_{low}, w_{high}]$와 도메인의 집합 D에 대한 이동 평균은 다음과 같이 정의된다.

$$MA(x) = \frac{1}{N} \sum_{x_i \in W, D} t_i \tag{12.6}$$

$$W = [x - w_{low}, x + w_{high}], \quad N = |\{x_i | x_i \in W, D\}| \tag{12.7}$$

그리고 time window가 각각$[-\frac{2\pi}{100} \times 10, \frac{2\pi}{100} \times 10]$, $[-\frac{2\pi}{100} \times 10, \frac{2\pi}{100} \times 10]$, $[-\frac{2\pi}{100} \times 20, \frac{2\pi}{100} \times 20]$와 같은 값을 가질 때 추정한 잠재 함수는 다음과 같다.

그림 9, 10, 11에서 살펴볼 수 있 듯, time window W의 크기에 따라 추정된

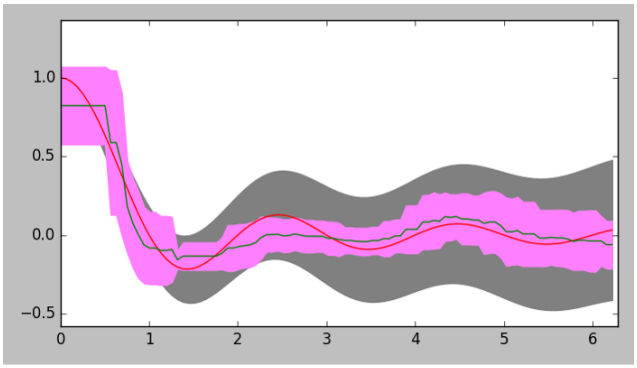


그림 12-9: time window $W = \left[-\frac{2\pi}{100} \times 10, \frac{2\pi}{100} \times 10\right]$ 일 때 추정된 잠재함수

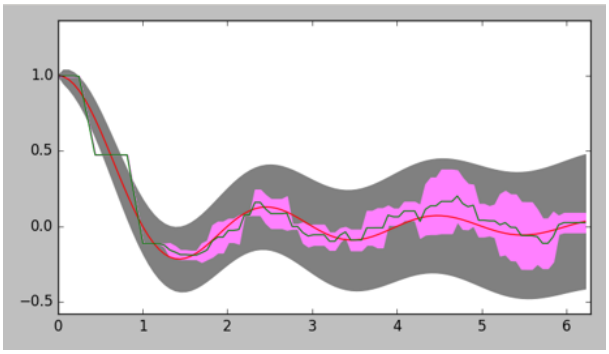


그림 12-10: time window $W = \left[-\frac{2\pi}{100} \times 3, \frac{2\pi}{100} \times 3\right]$ 일 때 추정된 잠재함수

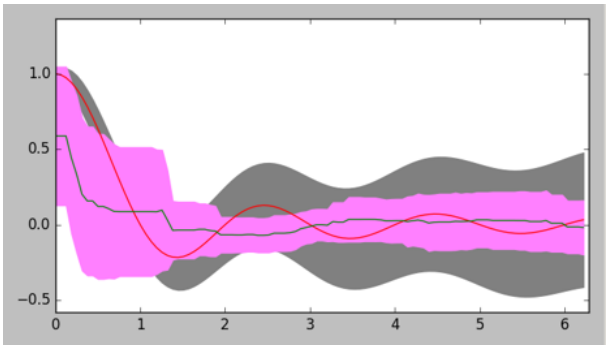


그림 12-11: time window $W = \left[-\frac{2\pi}{100} \times 20, \frac{2\pi}{100} \times 20\right]$ 일 때 추정된 잠재함수

잠재 함수가 다르며 여기서 적절한 time window W가 무엇인지 판별하기 어렵다. 또한 이동평균식에서의 가중치는 모두 1로 W 안에 있는 관측값이라면 하루 전의 관측값이든 10년 전의 관측값이든 같은 가중치로 평균을 구한다는 점 또한 이동 평균 방식이 가지는 한계점이다.

1.5 Simple Analyses with Differentiated Domain Correlation

앞의 문제를 해결하기 위해 이동 평균을 낼 때의 t_i의 가중치를 1이 아닌 x와 x_i의 거리에 따라 다르게 조절하였다.
먼저 그림 12와 같이 가중치가 선형적인 경우, 이를 바탕으로 추정한 잠재함수는 실제 값을 이전보다 더 잘 추정한다. 그러나 선형가중치의 경우에는 미분값을 구할 수 없는 지점이 존재하며 구현하는데 구간에 따라 그래프를 설정해주어야 하는 한계를 지닌다.

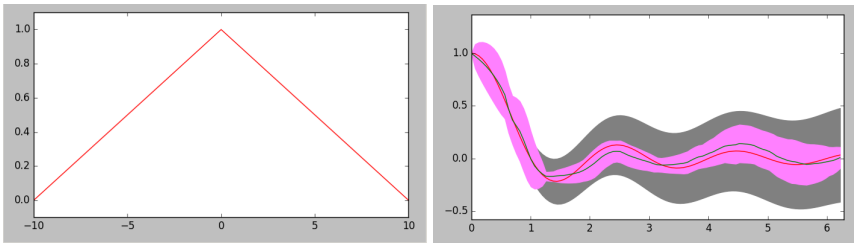


그림 12-12: 선형가중치 및 선형가중치로 추정한 잠재함수

이러한 문제점을 해결하기 위해 그림 12와 같이 squared exponential 함수를 사용하여 잠재함수를 추정한다.

squared exponential 함수는 연속 도메인 위의 한 x점 x_i와 에 대해 커널 함수로 $k(x, x_i) = exp\left(-\frac{|x-x_i|^2}{L^2}\right)$을 가진다. 이를 활용하여 도메인의 집합 D에 대한 이동 평균을 다음과 같이 정의할 수 있다.

$$MA(x) = \frac{1}{\sum_{x_i \in D} k(x, x_i)} \sum_{x_i \in D} k(x, x_i) t_i \tag{12.8}$$

여기서 time window가 식에 나타나지 않는 이유는 커널 함수에 의해 x와 x_i 사이의 거리가 L보다 커질 경우 값이 0으로 정의된다. 즉, 커널함수의 정의 속에 time window가 숨어있다고 볼 수 있다.

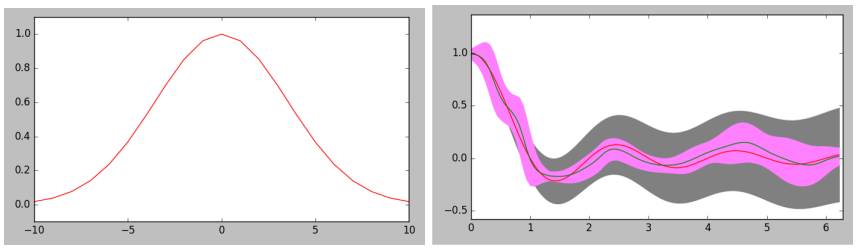


그림 12-13: 선형가중치 및 선형가중치로 추정한 잠재함수

2 Derivation of Gaussian Process Regression

2.1 Multivariate normal distribution

지금까지 관찰된 데이터셋을 바탕으로 잠재함수를 추정하는 몇가지 방법을 알아보았다. 가우시안 프로세스 리그레션 역시 이와 같은 맥락이다. 먼저, 가우시안 프로세스란 연속 도메인 위의 N개의 포인트들의 집합 $X_N = [x_1, x_2, \cdots, x_N]^T$에 대해 관찰된 포인트 집합을 $T_N = [t_1, t_2, \cdots, t_N]^T$라 할 때, T_N이 다변수정규분포를 따르는 것을 의미한다. 여기서 가우시안 프로세스 리그레션은 주어진 T_N을 바탕으로 다음에 관찰될 값 t_{n+1}을 유추하는 것이 목표이다. 이 목표는 T_N을 바탕으로 t_{N+1}의 확률분포인 $P(t_{N+1}|T_N)$을 찾음으로써 달성 될 수 있다.

따라서 가우시안 프로세스 리그레션에 대해 알아보기 전에, 그 바탕이 되는 다변수 정규분포에 대해 먼저 알아보겠다. k차원의 랜덤 벡터 $X = [X_1, X_2, \cdots, X_k]^T$가 다변수 정규분포를 따를 때 평균벡터, 공분산 행렬, 확률밀도 함수는 다음과 같이 표현된다.

$$X \sim N(\mu, \Sigma) \tag{12.9}$$

X가 위와 같은 분포일 때 k차원의 평균 벡터는 다음과 같다.

$$\mu = E[X] = [E[X_1], E[X_2], \cdots, E[X_k]]^T \tag{12.10}$$

$k \times k$ 차원의 공분산 행렬은 다음과 같이 나타난다.

$$\Sigma =: E\left[(X-\mu)(X=\mu)^T\right] = [Cov[X_i, X_j; 1 \leq i, j \leq l.]] \tag{12.11}$$

마지막으로 확률밀도함수(PDF)는 다음과 같다.

$$P(X) = \frac{1}{(2\pi)^{d/2}} \frac{1}{|\Sigma|^{1/2}} exp\left(-\frac{1}{2}(X-\mu)^T\Sigma^{-1}(X-\mu)\right) \tag{12.12}$$

추가적으로, 다변수 정규분포는 다음과 같은 성질을 따른다.

랜덤 벡터 $X = [X_1 \quad X_2]^T$가 $X \sim N(\mu, \Sigma)$을 따르고,

$\mu = [\mu_1 \quad \mu_2]^T$, $\Sigma = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}$ 일 때,

$$P(X_1) = N(X_1|\mu_1, \quad \Sigma_{11}) \tag{12.13}$$

$$P(X_1|X_2) = N(X_1|\mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(X_1 - \mu_2), \quad \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}) \tag{12.14}$$

위 두 공식은 앞으로 다변수 정규분포를 다루는데 유용하게 쓰일 것이다.

2.2 Linear Regression with Basis Function

이전 단원에서 잠재함수 Y를 유추해보았다면, 이번 단원에서는 Y의 확률분포를 구해보겠다. 연속 도메인 위의 N개의 포인트들의 집합 $X_N = [x_1, x_2, \cdots, x_N]^T$이 M차원으로 매핑되며 그 때의 M차원의 웨이트 벡터를 W, 매핑된 값을 $Y_N = [y_1, y_2, \cdots, y_N]^T$라고 했을 때, Y_N는 다음과 같은 수식으로 나타낼 수 있다.

$$y(x_i) = W^T\phi(x_i) \tag{12.15}$$

$$W = [w_1, w_2, \cdots, w_M]^T, \quad \phi(x_i) = [\phi_1(x_i), \phi_2(x_i), \cdots, \phi_M(x_i)]^T \tag{12.16}$$

모든 원소 $(x_1, y_1), (x_1, y_1), \cdots, (x_N, y_n)$에 대해 식 15를 행렬곱으로 표현하면 아래와 같이 표현된다.

$$\begin{aligned} Y &= \Phi W \\ &= \begin{bmatrix} \phi(x_1)^T \\ \phi(x_2)^T \\ \vdots \\ \phi(x_N)^T \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_M \end{bmatrix} \\ &= \begin{bmatrix} \phi_1(x_1) & \phi_2(x_1) & \cdots & \phi_M(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \cdots & \phi_M(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_1(x_N) & \phi_2(x_1) & \cdots & \phi_M(x_1) \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_M \end{bmatrix} \end{aligned} \tag{12.17}$$

전에는 W를 고정된 값으로 가정했지만, 이제는 이를 확률적으로 분포된 값으로 가정하겠다. W의 확률분포는 다음과 같이 정의한다.

$$P(W) = N(W|0, \alpha^{-1}I) \tag{12.18}$$

W와 Y는 식 15와 같이 표현되므로, W의 확률분포를 이용해서 Y의 확률분포를 구할 수 있다.

$$E[Y] = E[\Phi W] = \Phi E[W] \tag{12.19}$$

$$cov[Y] = E[(Y-0)(Y-0)^T] = E[YY^T] = E[\Phi W W^T \Phi^T] = \Phi E[WW^T]\Phi = \frac{1}{\alpha}\Phi\Phi^T \tag{12.20}$$

공분산 행렬의 원소를 다음과 같은 커널함수로 정의하면 다음과 같이 나타난다.

$$K_{nm} = k(x_n, x_m) = \frac{1}{\alpha}\phi(x_n)\phi(x_m) \tag{12.21}$$

따라서, Y의 확률 분포를 정리하면 다음과 같다.

$$P(Y) = N(W|0, K) \tag{12.22}$$

2.3 Modeling Noise With Gaussian Distribution

2.2에서 Y의 확률분포에 대해 알아보았다. 그렇다면 Y는 $P(t_{N+1}|T_N)$을 유추하는데 어떤 역할을 하는가? 연속 도메인의 하나의 포인트 x_n에 대해 실제로 관측된 값을 t_n, 가우시안 에러를 e_n이라고 하면 변수들끼리 다음과 같은 관계가 성립한다.

$$t_n = y_n + e_n \tag{12.23}$$

정규분포의 성질에 따라 조건부 확률 분포는 다음과 같다.

$$P(t_n|y_n) = N(t_n|y_n, \quad \beta^{-1}) \tag{12.24}$$

이를 모든 원소들에 대한 행렬로 나타내면 아래와 같다.

$$P(T|Y) = N(T|Y, \quad \beta^{-1}I_N) \tag{12.25}$$

2.4 Marginal Gaussian Distribution

조건 분포 $P(T|Y)$를 바탕으로 $P(T)$를 구할 수 있다. 일반적으로는 T와 Y의 결합 분포를 Y에 대해 적분하여 구할 수 있다.

$$P(T) = \int P(T|Y)P(Y)dY = \int N(T|Y, \beta^{-1}I_N)N(Y|0, K)dY \tag{12.26}$$

하지만 우리는 2.1에서 나왔던 다변수 정규분표의 성질을 이용하여 구해보도록 하겠다. 먼저 $P(T|Y)P(Y) = P(T, Y) = P(Z)$라고 정의하고 자연로그를 씌워서 간단히 하면 다음과 같다.

$$\begin{aligned}
\ln P(Z) &= \ln P(Y) + \ln P(T|Y) \\
&= -\frac{1}{2}(Y-0)^T K^{-1}(Y-0) - \frac{1}{2}(T-Y)^T \beta I_N (T-Y) + const \\
&= -\frac{1}{2}(Y)^T K^{-1}(Y) - \frac{1}{2}(T-Y)^T \beta I_N (T-Y) + const
\end{aligned} \tag{12.27}$$

상수를 제외하고 간단히 하면 다음과 같이 나타난다.

$$\begin{aligned}
\text{given equation} &= \ln P(Y) + \ln P(T|Y) \\
&= -\frac{1}{2}Y^T K^{-1} Y - \frac{\beta}{2}T^T T + \frac{\beta}{2}TY + \frac{\beta}{2}YT - \frac{\beta}{2}Y^T T \\
&= -\frac{1}{2}\begin{pmatrix} Y \\ T \end{pmatrix}^T \begin{pmatrix} K^{-1} + \beta I_N & -\beta I_N \\ -\beta I_N & \beta I_N \end{pmatrix} \begin{pmatrix} Y \\ T \end{pmatrix} \\
&= -\frac{1}{2}Z^T R Z
\end{aligned} \tag{12.28}$$

그런데 이 형태는 2.1에서 보았던 다변수 정규분포의 확률밀도함수(PDF)에 자연로그를 씌운 형태와 같다. 따라서 Z 는 평균 벡터가 0벡터, 공분산 행렬을 R^{-1}로 갖는 다변수 정규분포를 따른다.
공분산 행렬인 R^{-1}를 구할 때, 부분행렬로 이루어진 행렬의 역행렬을 구하는 아래의 공식을 이용할 수 있다.

$$|\begin{pmatrix} A & B \\ C & D \end{pmatrix}^{-1} = \begin{pmatrix} M & -MBD^{-1} \\ -D^{-1}CM & D^{-1} + D^{-1}CMBD^{-1} \end{pmatrix} \tag{12.29}$$

$$M = (A - BD^{-1}C)^{-1} \tag{12.30}$$

이를 이용하여 R^{-1}을 계산하면 다음과 같다.

$$M = (K^{-1} + \beta I_N - \beta I_N(\beta I_N)^{-1}\beta I_N)^{-1} = K \tag{12.31}$$

$$\begin{aligned} R^{-1} &= \begin{pmatrix} K & K\beta I_N(\beta I_N)^{-1} \\ (\beta I_N)^{-1}\beta I_N K & (\beta I_N)^{-1} + (\beta I_N)^{-1}\beta I_N K \beta I_N (\beta I_N)^{-1} \end{pmatrix} \\ &= \begin{pmatrix} K & K \\ K & (\beta I_N)^{-1} + K \end{pmatrix} \end{aligned} \tag{12.32}$$

따라서 Z의 확률분포는 다음과 같이 나타난다.

$$P(Z) = N\left(Z|0, \begin{pmatrix} K & K \\ K & (\beta I_N)^{-1} + K \end{pmatrix}\right) \tag{12.33}$$

위 식을 마탕으로 2.1에서 보았던 다변수 정규분포의 성질 식 12.13을 이용하여 결합분포$P(Y,T)$를 바탕으로 단일분포$P(T)$를 쉽게 구할 수 있다.

$$P(T) = N\left(T|0, (\beta I_N)^{-1} + K\right) \tag{12.34}$$

2.5 Mean and Covariance of $P(t_{N+1}|T_N)$

2.4에서 구한 것처럼 $P(T) = N\left(T|0, (\beta I_N)^{-1} + K\right)$임으로 $T_N = [t_1, t_2, \cdots, t_N]^T$에 새로 관측할 포인트 t_{N+1}을 추가한 벡터 T_{N+1}의 확률분포는 다음과 같다.

$$\begin{aligned} P(T_{N+1}) &= P(T_N, t_{N+1}) \\ &= N\left(T|0, \begin{pmatrix} (\beta I_N)^{-1} + K & k_{1(N+1)} \\ K_{(N+1)1} & K_{(N+1)(N+1)} + \beta \end{pmatrix}\right) \end{aligned} \tag{12.35}$$

$$cov_{N+1} = \begin{bmatrix} cov_N & k \\ K^T & c \end{bmatrix} \tag{12.36}$$

또한 2.1에서 언급했던 식 12.14을 이용하여 결합분포를 알 때, 조건 분포를

쉽게 구할 수 있다.

$$P(t_{N+1}|T_N) = N\left(t_{N+1}|0 + k^T cov_N^{-1}(T_N - 0), c - k^T cov_N^{-1}k\right) \tag{12.37}$$

따라서 $\mu_{t_{N+1}}$과 $\sigma_{t_{N+1}}$은 $\mu_{t_{N+1}|T_N} = k^T cov_N^{-1} T_N$, $\sigma_{t_{N+1}|T_N} = c - k^T cov_N^{-1} k$ 와 같이 나타난다.

2.6 Hyperparameters of Gaussian Process Regression

지금까지 가우시안 프로세스 리그레션이 어떻게 유도되는지 살펴 보았다. 이제는 각각 다른 하이퍼파라미터의 값을 가진 가우시안 프로세스가 어떻게 회귀되는지 살펴보겠다.

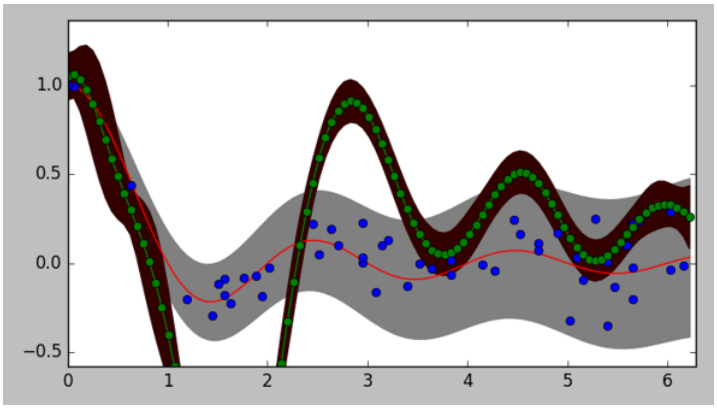


그림 12-14: $(\theta_0, \theta_0, \theta_0, \theta_0, \beta) = (0.5, 10, 0, 0, 0.01)$ 일 때의 가우시안 프로세스 리그레션

위 그림을 통해 가우시안 프로세스 리그레션은 하이퍼 파라미터의 영향을 많이 받음을 알 수 있다. 즉, 하이퍼 파라미터를 학습시키지 않는다면 좋은 추정을 할 수 없다. 적합한 학습을 거친다면 다음과 같은 결과를 얻을 수 있다.

2.7 Probabilistic Programming for Hyperparameter Learning of GP

다음과 같은 커널함수를 사용하여 커널 하이퍼파라미터를 텐서플로우라는 프레임워크를 이용하여 학습시켜 보겠다.

$$K_{nm} = k(x_n, x_m) = \theta_0 exp\left(-\frac{\theta_1}{2} \parallel x_n - x_m \parallel^2\right) + \theta_2 + \theta_3 x_n^T x_m \tag{12.38}$$

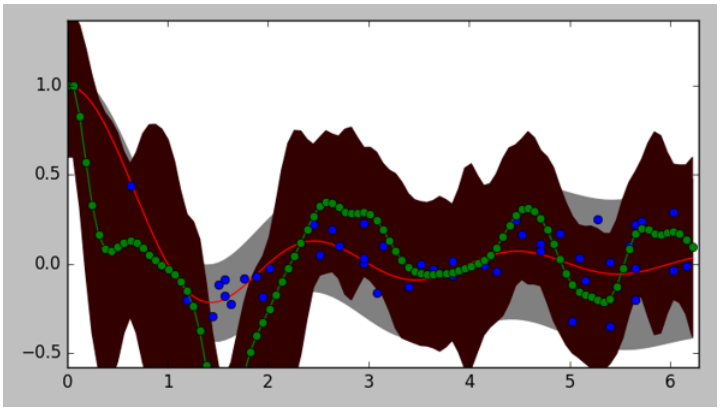


그림 12-15: $(\theta_0, \theta_0, \theta_0, \theta_0, \beta) = (0.5, 50, 0, 0, 0.01)$ 일 때의 가우시안 프로세스 리그레션

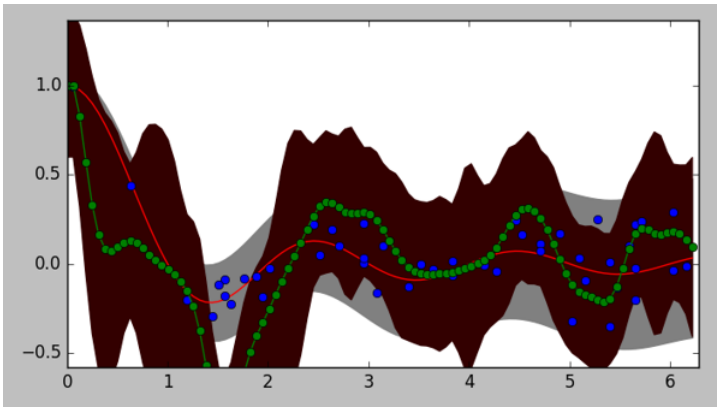


그림 12-16: $(\theta_0, \theta_0, \theta_0, \theta_0, \beta) = (1.377, 1.22, 1.354, 1.858, 9.980)$ 일 때의 가우시안 프로세스 리그레션

이러한 커널함수를 'KernelFunctionWithTensorFlow'로 정의하였다.

```
def KernelFunctionWithTensorFlow(theta0, theta1, theta2, theta3, X1, X2):
    insideExp = tf.multiply(tf.div(theta1, 2.0), tf.matmul((X1 - X2), tf.transpose(X1 - X2)))
    firstTerm = tf.multiply(theta0, tf.exp(-insideExp))
    secondTerm = theta2
    thridTerm = tf.multiply(theta3, tf.matmul(X1, tf.transpose(X2)))
    ret = tf.add(tf.add(firstTerm, secondTerm), thridTerm)
    return ret
```

그림 12-17: 커널함수를 KernelFunctionWithTensorFlow로 정의

그림 18의 코드는 Covariance Matrix를 구성하는 과정을 포함하고 있으며, 그림 19의 코드는 관측된 데이터와 앞에서 구한 Covariance Matrix를 기반으로 다음 데이터 값을 예측하는 과정을 담고 있다.

```
def KernelHyperParameterLearning(trainingX, trainingY):
    tf.reset_default_graph()
    numDataPoints = len(trainingY)
    numDimension = len(trainingX[0])

    # Input and Output Data Declaration for Tensorflow
    obsX = tf.placeholder(tf.float32, [numDataPoints, numDimension])
    obsY = tf.placeholder(tf.float32, [numDataPoints, 1])

    # Learning Parameter Variable Declaration for TensorFlow
    theta0 = tf.Variable(1.0)
    theta1 = tf.Variable(1.0)
    theta2 = tf.Variable(1.0)
    theta3 = tf.Variable(1.0)
    beta = tf.Variable(1.0)

    # Kernel Bulid
    matCovarianceLinear = []
    for i in range(numDataPoints):
        for j in range(numDataPoints):
            kernelEvaluationResult = KernelFunctionWithTensorFlow(theta0, theta1, theta2, theta3,
                                                                  tf.slice(obsX, [i, 0], [1, numDimension]),
                                                                  tf.slice(obsX, [j, 0], [1, numDimension]))
            if i != j:
                matCovarianceLinear.append(kernelEvaluationResult)
            if i == j:
                matCovarianceLinear.append(kernelEvaluationResult + tf.div(1.0, beta))

    matCovarianceCombined = tf.convert_to_tensor(matCovarianceLinear, dtype=tf.float32)
    matCovariance = tf.reshape(matCovarianceCombined, [numDataPoints, numDataPoints])
    matCovarianceInv = tf.matrix_inverse(matCovariance)
```

그림 12-18: 하이퍼파라미터 학습 구현1

```
# Prediction
sumsquarederror = 0.0
for i in range(numDataPoints):
    k = tf.Variable(tf.ones([numDataPoints]))
    for j in range(numDataPoints):
        kernelEvaluationResult = KernelFunctionWithTensorFlow(theta0, theta1, theta2, theta3,
                                                              tf.slice(obsX, [i, 0], [1, numDimension]),
                                                              tf.slice(obsX, [j, 0], [1, numDimension]))
        indices = tf.constant([j])
        tempTensor = tf.Variable(tf.zeros([1]))
        tempTensor = tf.add(tempTensor, kernelEvaluationResult)
        tf.scatter_update(k, tf.reshape(indices, [1, 1]), tempTensor)

    c = tf.Variable(tf.zeros([1, 1]))
    kernelEvaluationResult = KernelFunctionWithTensorFlow(theta0, theta1, theta2, theta3,
                                                          tf.slice(obsX, [i, 0], [1, numDimension]),
                                                          tf.slice(obsX, [i, 0], [1, numDimension]))

    c = tf.add(tf.add(c, kernelEvaluationResult), tf.div(1.0, beta))

    k = tf.reshape(k, [1, numDataPoints])

    predictionMu = tf.matmul(k, tf.matmul(matCovarianceInv, obsY))
    predictionVar = tf.subtract(c, tf.matmul(k, tf.matmul(matCovarianceInv, tf.transpose(k))))

    sumsquarederror = tf.add(sumsquarederror, tf.pow(tf.subtract(predictionMu, tf.slice(obsY, [i, 0], [1, 1])), 2))

# Training session declaration
training = tf.train.GradientDescentOptimizer(0.1).minimize(sumsquarederror)
```

그림 12-19: 하이퍼파라미터 학습 구현2

2.8 Bayesian Optimization with Gaussian Process

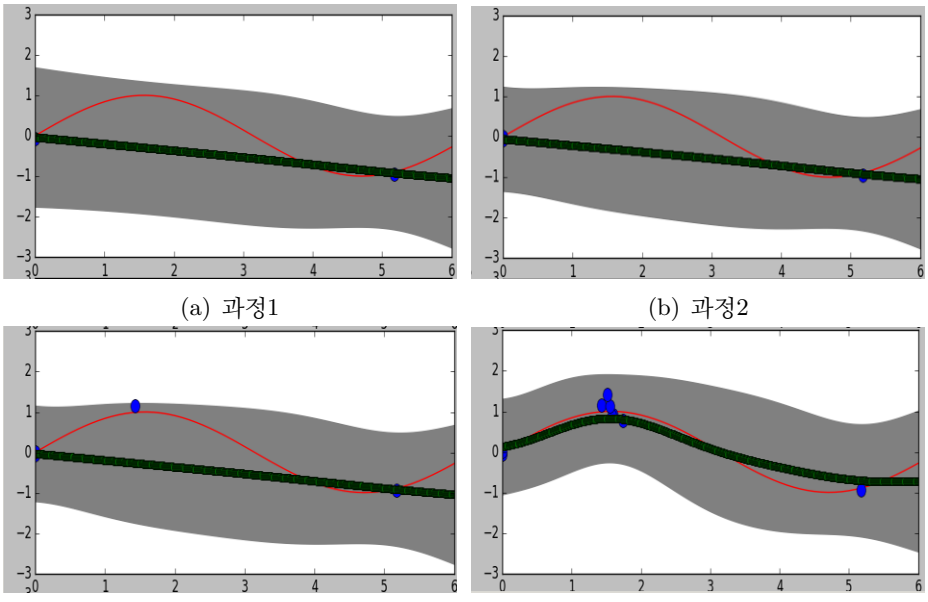
지금까지 가우시안 프로세스 리그레션에 대해 알아보았다. 이제부터는 가우시안 프로세스를 적용할 수 있는 베이지안 최적화에 대해 알아보겠다. 베이지안 최적화는 우리가 순차적이고 입력값을 정할 수 있는 실험을 진행한다고 가정할 때, 결과값을 최대 혹은 최소로 만드는 입력값(input)을 찾는 것을 목표로 한다. 베이지안 최적화는 우리가 결과값을 결정하는 잠재함수를 모른다는 점, 결과값

과 입력값이 연속적이다는 점, 결과값이 확률적(stochastic)으로 발생한다는 점이 특징이다. 베이지안 최적화의 목적을 수식으로 나타내면 아래와 같다.

$$x^* = argmax_{x \in X} f(x) \tag{12.39}$$

이러한 문제를 풀기 위한 기존에 알려진 방법에는 'Grid Search'가 있다. 이 방법은 도메인을 적당한 격자로 나누어 각각의 실험값을 찾아냄으로써 가장 큰 결과값을 찾아내는 방법이다. 다른 방법으로는 'Binary Search'가 있다. 이 방법은 하나의 실험값에 대해 양 옆의 실험값을 알아내어 결과값이 더 큰 방향으로 동일 과정을 진행하는 방법이다. 그러나 이 과정들은 잠재함수를 학습하여 최적값을 찾아내는 방식이 아니다.

베이지안 최적화는 잠재함수에 대해 학습하는 과정과 다음 샘플링 입력값을 선택하는 과정이 번갈아가며 일어나는 과정이다. 베이시안 최적화는 아래 4가지 연속되는 그림과 같이 이루어진다. 과정1-3에서는 분산을 줄여주게끔 샘플링하는 exploitation 과정을, 과정4에서는 평균이 커지는 지점을 반복하여 샘플링하는 exploration 과정을 수행하며 과정인 진행될수록 잠재함수가 실제함수와 유사하게 학습되는 것을 확인할 수 있다.



(a) 과정1 (b) 과정2

(c) 과정3 (d) 과정4

그림 12-20: 베이시안 최적화 과정

2.9 Acquisition Function: Maximum Probability of Improvement

이전 단원에서 exploitation과 exploration 사이에서 적절한 샘플링 방식을 선택하여 베이시안 최적화를 진행한다고 하였다. 여기서 적절한 샘플링 방식을 결정해주는 함수가 'Acquisition Function'이다. 가우시안 프로세스는 어느 포인트에서라도 예측된 평균과 표준편차값을 알려준다. 여기서 예측된 평균은 우리가 최적화된 값을 찾을 수 있다는 의미이고, 예측된 표준편차값은 예측의 리스크를 의미한다. 'Acquistion Function'은 이 두 요소를 고려하여 다음 샘플링 포인트를 찾아준다.

이러한 Aquisition Function에는 여러가지가 있는데 그 중에서 'Maximum Probability of Improvement'에 대해 알아보겠다. 'Maximum Probability of Improvement'는 기존의 데이터 D가 가지고 있는 함수 $f(x)$의 최대값을 y_{max}, 새로운 샘플링 포인트 x를 잡았을 때의 합수값 $y = f(x)$라고 할 때, y가 y_{max}보다 margin m 만큼 혹은 더 크게 할 확률 중, 가장 높은 확률을 가지게 하는 x를 구하는 방법이다. 유도는 다음과 같다.

$$y \sim N(\mu, \sigma^2), CDF function \Phi \tag{12.40}$$

$$\begin{aligned} MPI(x|D) &= argmax_x P(y \geq (1+m)y_{max}|x, D) \\ &= argmax_x P(\frac{y-\mu}{\sigma} \geq \frac{(1+m)y_{max}-\mu}{\sigma}) \\ &= argmax_x 1 - \Phi(\frac{(1+m)y_{max}-\mu}{\sigma}) \\ &= argmax_x \Phi(\frac{\mu-(1+m)y_{max}}{\sigma}) \end{aligned}$$

이 과정과 'Grid search'와의 차이점에 의문이 들 수 있다. 'Grid search'는 실제 실험을 진행하여 포인트들의 함수값을 실제로 계산하는 것이라면 'Maximum Probability of Improvement'는 실험을 수행하는 것이 아닌, 포인트들에 대한 누적분포값 $\Phi(\frac{\mu-(1+m)y_{max}}{\sigma})$만을 구하고 가장 높은 값을 가진 x를 다음 샘플링 포인트로 결정하는 과정이다.

2.10 Acquisition Function: Maximum Expected Improvement

다음으로 'Maximum Expected Improvement'에 대해 알아 보겠다. 이는 앞서 다뤘던 'Maximum Probability of Improvement'의 문제점을 보안한 것이다. 'Maximum Probability of Improvement'에서는 margin의 역할을 하는 하이퍼파라미터 m 역시 최적화 해주어야 한다. 따라서 'Maximum Expected Improvement'에서는 마진 m을 0에서 무한대까지 순차적으로 변화시킴으로써 m에 대한 평균값이 가장 큰 x를 구한다.

여기서는 다음과 같은 가정이 필요하다.

$$y = f(x), \quad y_{max} = \underset{m=1,\cdots,n}{max} f(x_n)$$

$$u = \frac{y_{max} - \mu}{\sigma}, \quad v = \frac{u - \mu}{\sigma}$$

$$\mu = f(x|D), \quad \sigma = K(x|D)$$

$$m = max(0, y - y_{max}) = max(0, (v - u)\sigma)$$

이 때, 아래와 같은 조건을 만족하는 x를 구한다.

$$MEI(x|D) = argmax_x \int_0^\infty P(y_{max} + m)mdm \tag{12.41}$$

이 식을 정리하면 아래와 같다.

$$
\begin{aligned}
\int_0^\infty P(y \geq y_{max} + m)mdm &= \int_0^\infty P(\frac{y-\mu}{\sigma} \geq \frac{y_{max}-\mu+m}{\sigma})mdm \\
&= \int_0^\infty P(v \geq u + \frac{m}{\sigma})mdm \\
&= \int_0^\infty \int_{u+\frac{m}{\sigma}}^\infty \phi(\tilde{v})d\tilde{v}mdm \\
&= \int_0^\infty \int_0^\infty \chi_{[u+\frac{m}{\sigma},\infty)}(\tilde{v})\phi(\tilde{v})d\tilde{v}mdm \\
&= \int_0^\infty \int_0^\infty m\chi_{[u+\frac{m}{\sigma},\infty)}(\tilde{v})\phi(\tilde{v})d\tilde{v}dm \\
&= \int_0^\infty \int_0^\infty m\chi_{[u+\frac{m}{\sigma},\infty)}(\tilde{v})\phi(\tilde{v})dmd\tilde{v} \\
&= \int_0^\infty \left\{ \int_0^\infty m\chi_{[u+\frac{m}{\sigma},\infty)}(\tilde{v})dm \right\}\phi(\tilde{v})dmd\tilde{v} \\
&= \int_0^\infty \left\{ \int_0^\infty m\chi_{\{0\leq m\leq\sigma(\tilde{v}-u)\}}(\tilde{v})dm \right\}\phi(\tilde{v})d\tilde{v} \\
&= \int_0^\infty \left\{ \int_0^\infty m\chi_{\{0\leq m\leq\sigma(\tilde{v}-u)\}\cap\{0\leq\sigma(\tilde{v}-u)\}}(\tilde{v})dm \right\}\phi(\tilde{v})d\tilde{v} \\
&= \int_0^\infty \left\{ \int_0^\infty m\chi_{\{0\leq m\leq\sigma(\tilde{v}-u)\}}(\tilde{v})\chi_{\{\cap\{0\leq\sigma(\tilde{v}-u)\}}(\tilde{v})dm \right\}\phi(\tilde{v})d\tilde{v} \\
&= \int_0^\infty \left\{ \int_0^\infty m\chi_{\{0\leq m\leq\sigma(\tilde{v}-u)\}}(\tilde{v})dm \right\}\chi_{\{0\leq\sigma(\tilde{v}-u)\}}(\tilde{v})\phi(\tilde{v})d\tilde{v} \\
&= \int_u^\infty \left\{ \int_0^{\sigma(\tilde{v}-u)} mdm \right\}\phi(\tilde{v})d\tilde{v} \\
&= \int_u^\infty \frac{1}{2}\sigma^2(\tilde{v}-u)^2\phi(\tilde{v})d\tilde{v} \\
&= \frac{1}{2}\sigma^2\left[\int_u^\infty \tilde{v}^2\phi(\tilde{v})d\tilde{v} - 2u\int_u^\infty \tilde{v}\phi(\tilde{v})d\tilde{v} + u^2\int_u^\infty \phi(\tilde{v})d\tilde{v} \right] \\
&= \frac{1}{2}\sigma^2\left[-u\phi(u) + (1+u^2)\Phi(-u) \right]
\end{aligned}
$$

아래의 그림은 베이지안 최적화 과정에서 Maximum Expected Improvement 로 샘플링을 진행했을 때 iteration에 따라 샘플링되어 학습된 함수와 Probability Improvment 함수와 Expected Improvment 함수를 나타낸 결과다.

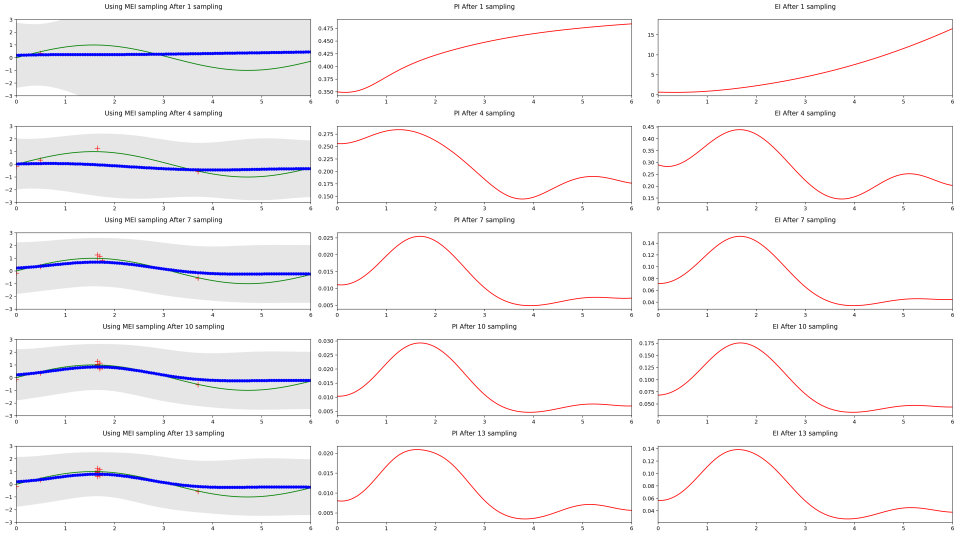


그림 12-21: MEI 샘플링 과정