Custom PyMC3 nonparametric Bayesian models built on top of the scikit-learn API

Bayesian Data Science DC Meetup

Daniel Emaasit
Data Scientist
Haystax Technology

October 11, 2018

Materials

Download slides & code:

bit.ly/pymc-learn-dc

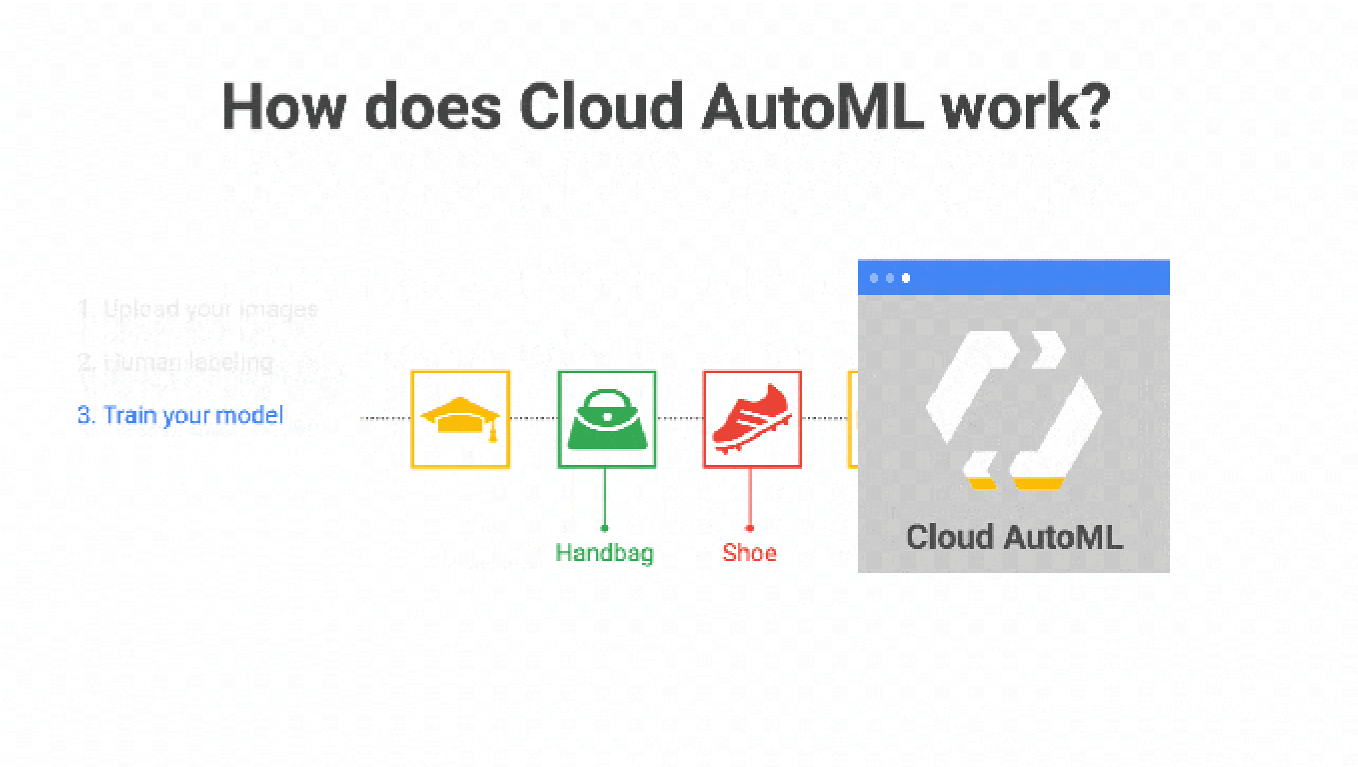
Application (1/3)

- Optimizing complex models in autonomous vehicles.

Schneider et al., 2016. (Uber ATG).

Application (2/3)

- Automating machine learning



TechCrunch, accessed 2018 (Google AutoML).

Application (3/3)

- Supplying internet to remote areas

GOOGLE'S INTERNET-BEAMING BALLOON GETS A NEW PILOT: AI

PROJECT LOON

Wired magazine, accessed 2016. (Google Project Loon)

Application (3/3)

- Supplying internet to remote areas

GOOGLE'S INTERNET-BEAMING BALLOON GETS A NEW PILOT: AI

PROJECT LOON

Wired magazine, accessed 2016. (Google Project Loon)

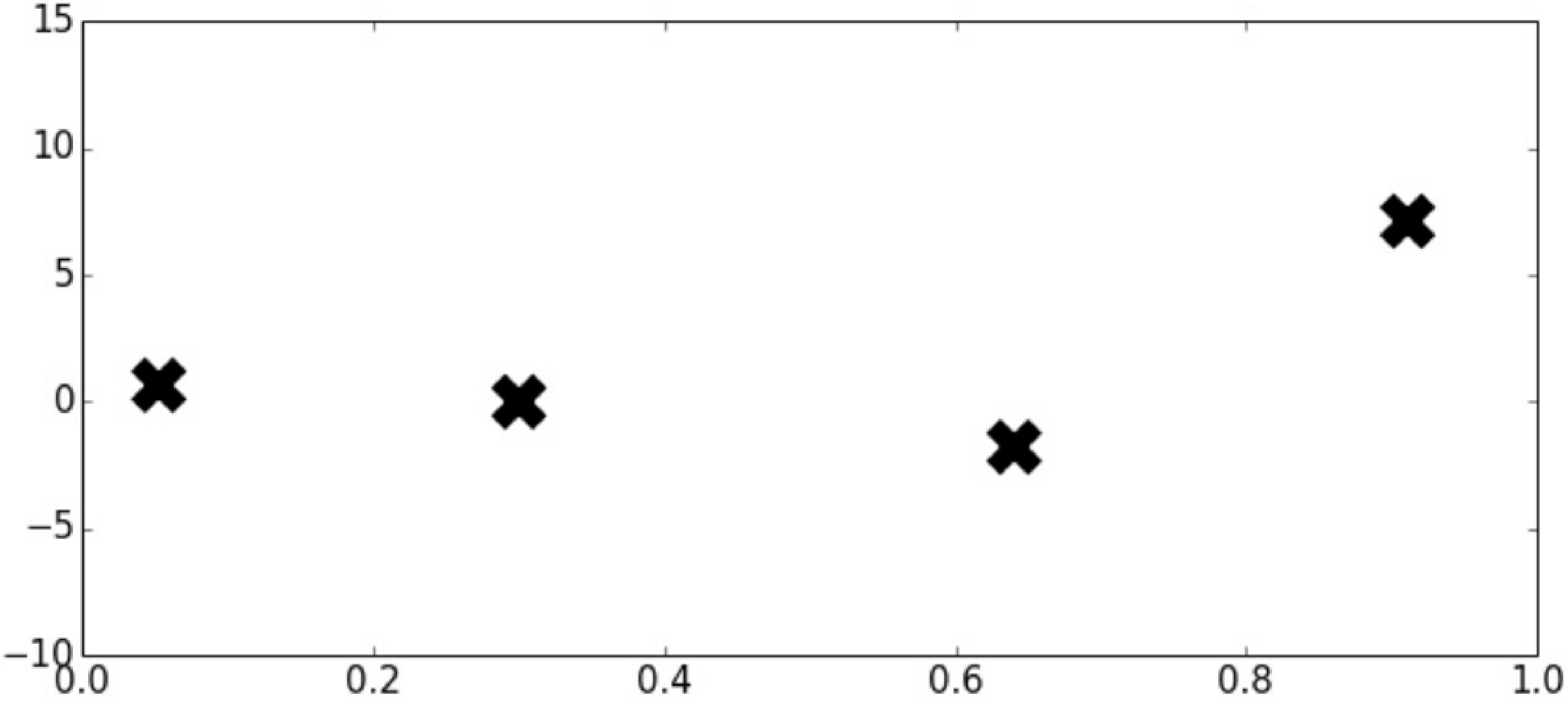
Intro to Bayesian Nonparametrics

Bayesian Nonparametrics (1/2)

- Function approximation.

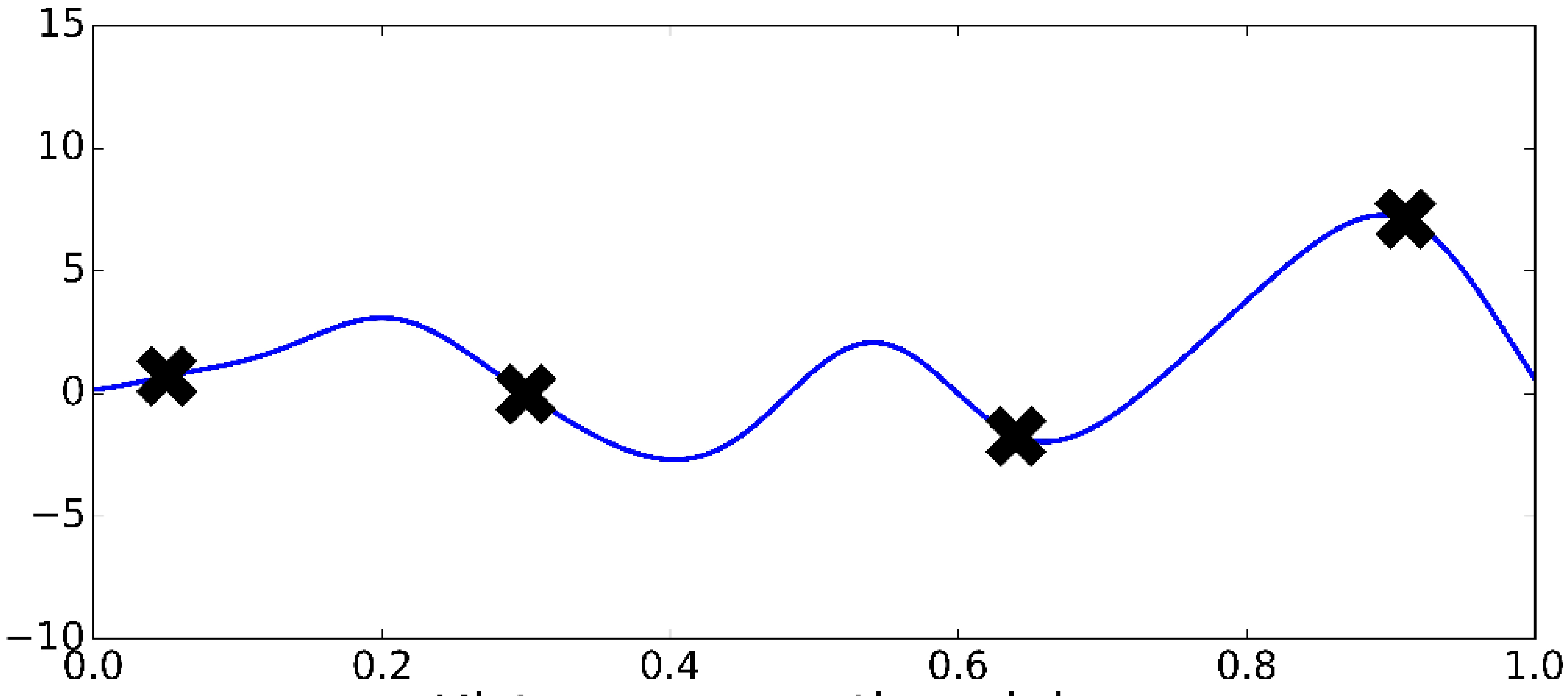
Bayesian Nonparametrics (1/2)

- Function approximation.



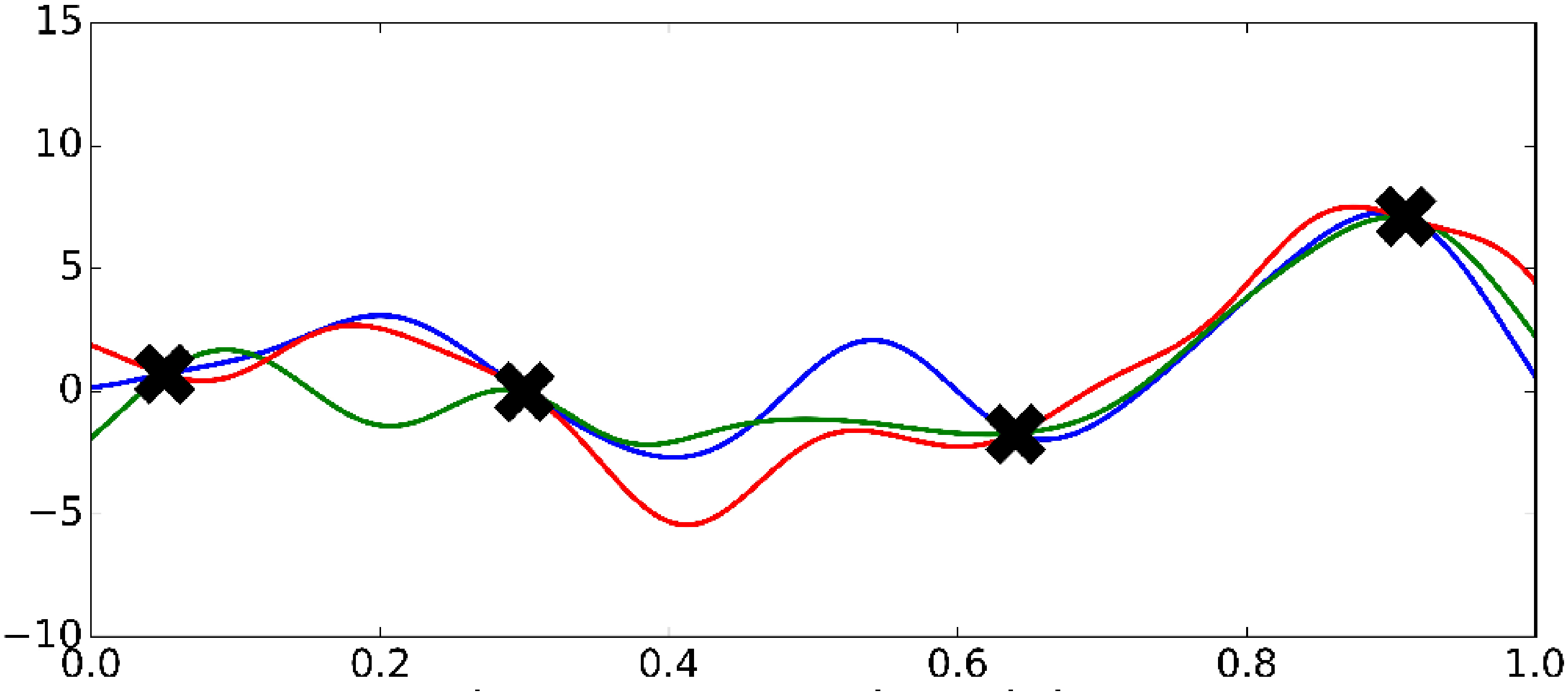
Bayesian Nonparametrics (1/2)

- Function approximation.



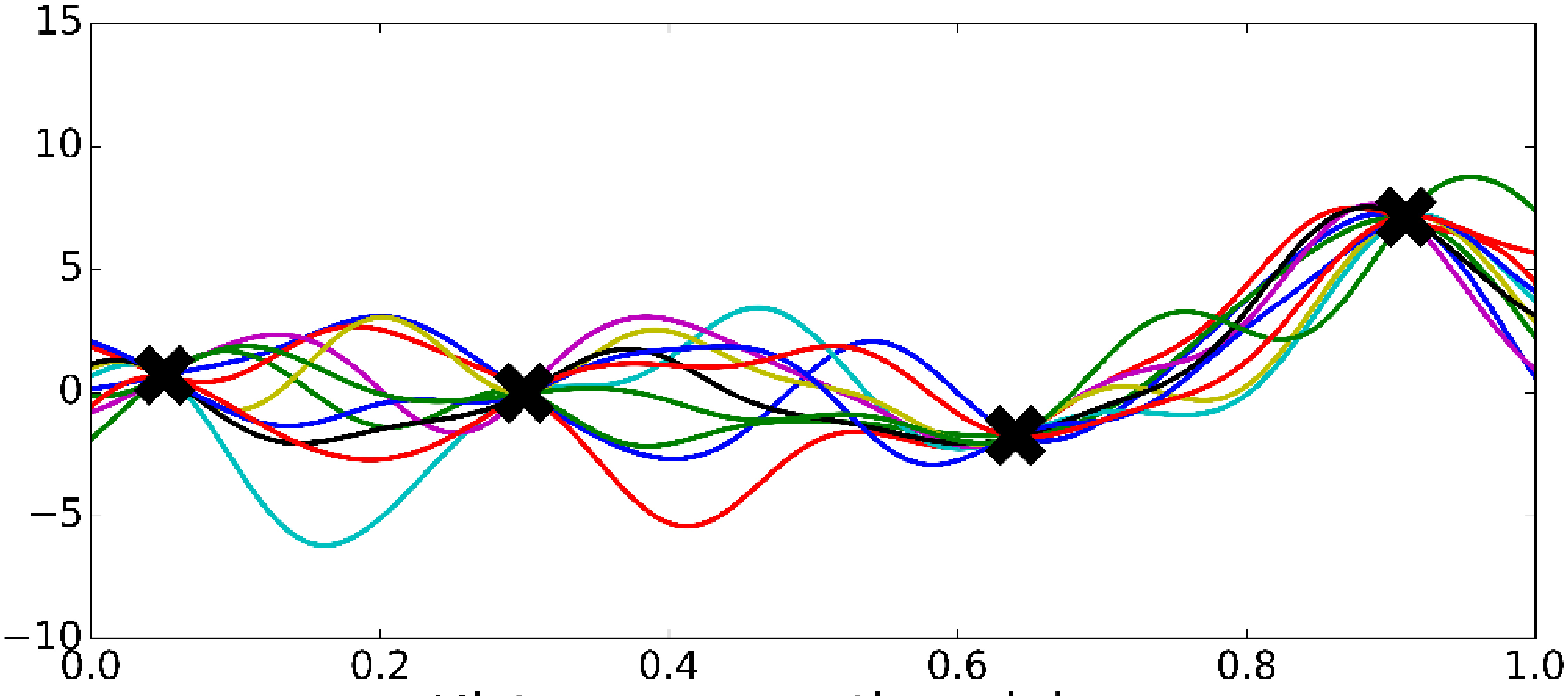
Bayesian Nonparametrics (1/2)

- Function approximation.



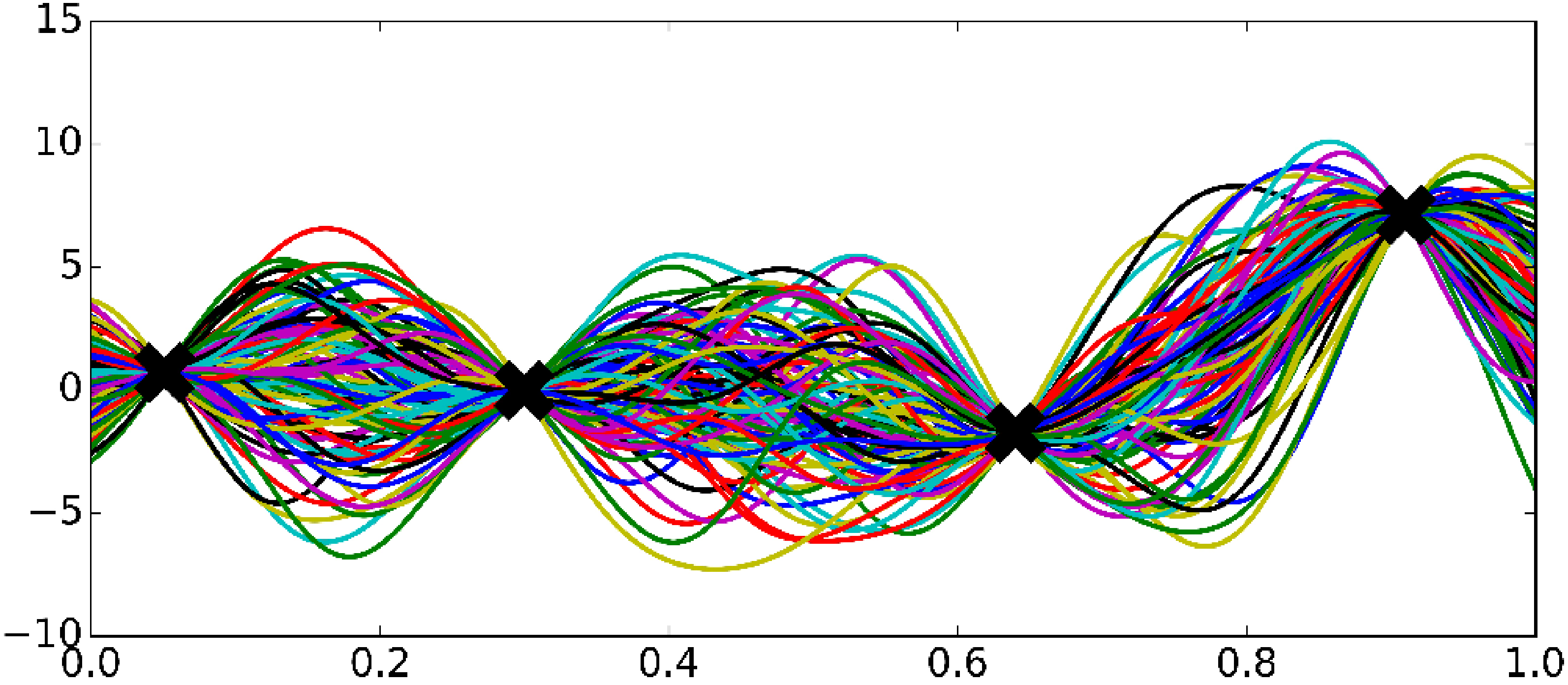
Bayesian Nonparametrics (1/2)

- Function approximation.



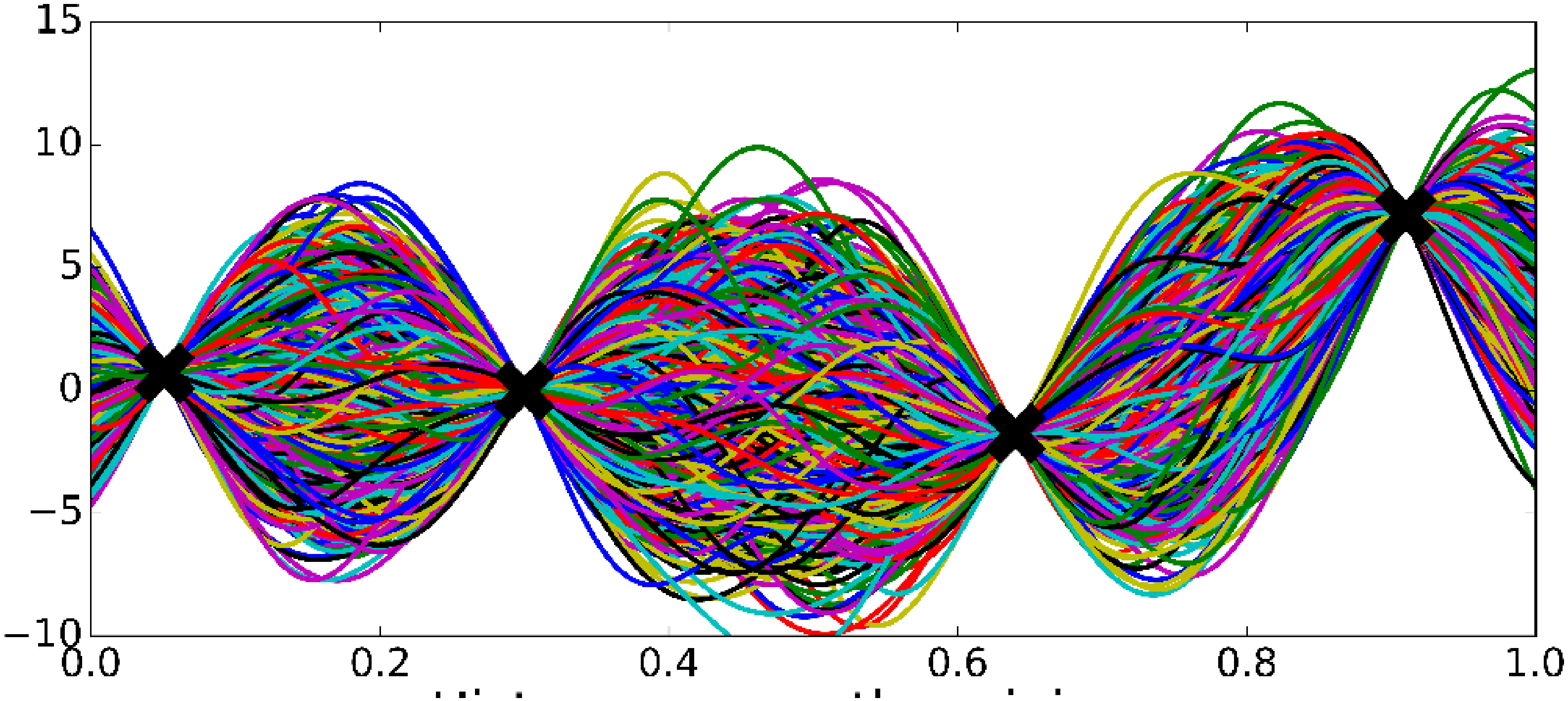
Bayesian Nonparametrics (1/2)

- Function approximation.



Bayesian Nonparametrics (1/2)

- Function approximation.



Bayesian Nonparametrics (2/2)

- Most approaches to modeling focus on **parametric models** that impose **restrictive** assumptions, e.g:

Bayesian Nonparametrics (2/2)

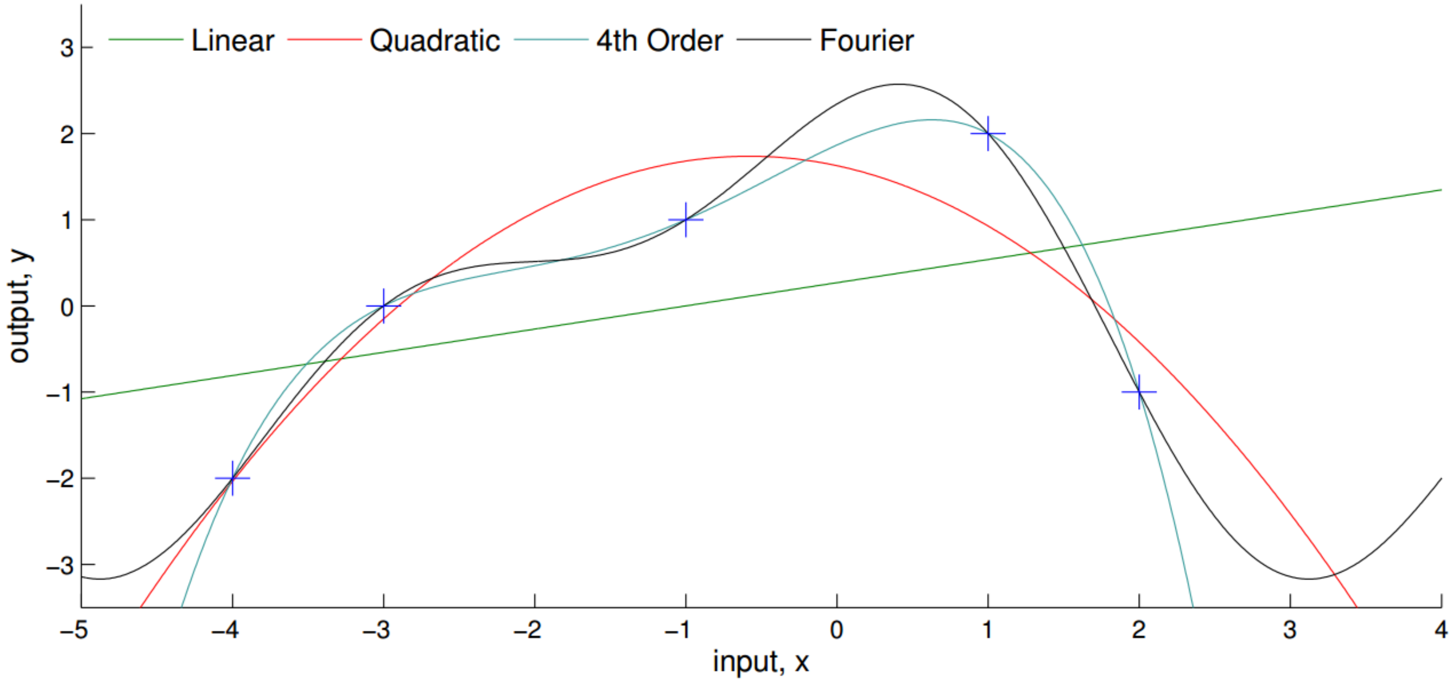
- Most approaches to modeling focus on **parametric models** that impose **restrictive** assumptions, e.g:

  1. **prespecifying the functional form,** $y = f(\mathbf{x}) + \varepsilon$

suppose y is continuous

$y_i \sim \mathbf{N}(\pi_i, \sigma^2)$

$\pi = \mathbf{E}[Y \mid X] = \beta_0 + \beta_1 x_1 + \beta_2 x_2$



Bayesian Nonparametrics (2/2)

- Most approaches to modeling focus on **parametric models** that impose **restrictive** assumptions, e.g:

1. **prespecifying the functional form,** $y = f(\mathbf{x}) + \varepsilon$

suppose y is continuous

$$y_i \sim \mathbf{N}(\pi_i, \sigma^2)$$

$$\pi = \mathbf{E}[Y \mid X] = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

Or

suppose y is discrete $\{0, 1\}$

$$y_i \sim Bin(n_i, \pi_i)$$

$$\pi = \mathbf{E}[Y \mid X] = \frac{e^{(\beta_0 + \beta_1 x_1 + \beta_2 x_2)}}{1 + e^{(\beta_0 + \beta_1 x_1 + \beta_2 x_2)}}$$

Bayesian Nonparametrics (2/2)

- Most approaches to modeling focus on **parametric models** that impose **restrictive** assumptions, e.g:

   1. **prespecifying the functional form,** $y = f(\mathbf{x}) + \varepsilon$

suppose y is continuous

$y_i \sim \mathbf{N}(\pi_i, \sigma^2)$

$\pi = \mathbf{E}[Y \mid X] = \beta_0 + \beta_1 x_1 + \beta_2 x_2$

Or

suppose y is discrete$\{0, 1\}$

$y_i \sim Bin(n_i, \pi_i)$

$\pi = \mathbf{E}[Y \mid X] = \frac{e^{(\beta_0 + \beta_1 x_1 + \beta_2 x_2)}}{1 + e^{(\beta_0 + \beta_1 x_1 + \beta_2 x_2)}}$

- Limitations of this approach:

   1. **it's difficult to know *a priori* the most appropriate function**

Bayesian Nonparametrics (2/2)

- Most approaches to modeling focus on **parametric models** that impose **restrictive** assumptions, e.g:

1. **prespecifying the functional form,** $y = f(\mathbf{x}) + \varepsilon$

suppose y is continuous

$$y_i \sim \mathbf{N}(\pi_i, \sigma^2)$$

$$\pi = \mathbf{E}[Y \mid X] = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

Or

suppose y is discrete$\{0, 1\}$

$$y_i \sim Bin(n_i, \pi_i)$$

$$\pi = \mathbf{E}[Y \mid X] = \frac{e^{(\beta_0 + \beta_1 x_1 + \beta_2 x_2)}}{1 + e^{(\beta_0 + \beta_1 x_1 + \beta_2 x_2)}}$$

2. **prespecifying the number of parameters, e.g** coefficients $\boldsymbol{\beta}$, variance $\boldsymbol{\sigma}^2$

$$\mathbf{E}[Y \mid X] = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$
$$\mathbf{E}[Y \mid X] = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2 + \beta_3 x_2^2 + \beta_4 sin(1 + x_2)$$

- Limitations of this approach:

1. **it's difficult to know *a priori* the most appropriate function**

Bayesian Nonparametrics (2/2)

- Most approaches to modeling focus on **parametric models** that impose **restrictive** assumptions, e.g:

  1. **prespecifying the functional form,** $y = f(\mathbf{x}) + \varepsilon$

suppose y is continuous

$y_i \sim \mathbf{N}(\pi_i, \sigma^2)$

$\pi = \mathbf{E}[Y \mid X] = \beta_0 + \beta_1 x_1 + \beta_2 x_2$

Or

suppose y is discrete$\{0,1\}$

$y_i \sim Bin(n_i, \pi_i)$

$\pi = \mathbf{E}[Y \mid X] = \frac{e^{(\beta_0+\beta_1 x_1+\beta_2 x_2)}}{1+e^{(\beta_0+\beta_1 x_1+\beta_2 x_2)}}$

  2. **prespecifying the number of parameters, e.g** $\boldsymbol{\text{coefficients } \beta, \text{variance } \sigma^2}$

$$\mathbf{E}[Y \mid X] = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$
$$\mathbf{E}[Y \mid X] = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2 + \beta_3 x_2^2 + \beta_4 sin(1 + x_2)$$

- Limitations of this approach:

  1. **it's difficult to know *a priori* the most appropriate function**
  2. **it's difficult to know *a priori* the most appropriate number of parameters**

Bayesian Nonparametrics (2/2)

- Most approaches to modeling focus on **parametric models** that impose **restrictive** assumptions, e.g:

1. **prespecifying the functional form,** $y = f(\mathbf{x}) + \varepsilon$

suppose y is continuous

$$y_i \sim \mathbf{N}(\pi_i, \sigma^2)$$

$$\pi = \mathbf{E}[Y \mid X] = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

Or

suppose y is discrete$\{0, 1\}$

$$y_i \sim Bin(n_i, \pi_i)$$

$$\pi = \mathbf{E}[Y \mid X] = \frac{e^{(\beta_0 + \beta_1 x_1 + \beta_2 x_2)}}{1 + e^{(\beta_0 + \beta_1 x_1 + \beta_2 x_2)}}$$

2. **prespecifying the number of parameters, e.g** coefficients $\boldsymbol{\beta}$, variance $\boldsymbol{\sigma}^2$

$$\mathbf{E}[Y \mid X] = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$
$$\mathbf{E}[Y \mid X] = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2 + \beta_3 x_2^2 + \beta_4 sin(1 + x_2)$$

- Limitations of this approach:

1. **it's difficult to know *a priori* the most appropriate function**
2. **it's difficult to know *a priori* the most appropriate number of parameters**
3. **pre-specifying** $f(\mathbf{X})$ **may produce either overly complex or simple models**

Gaussian Process

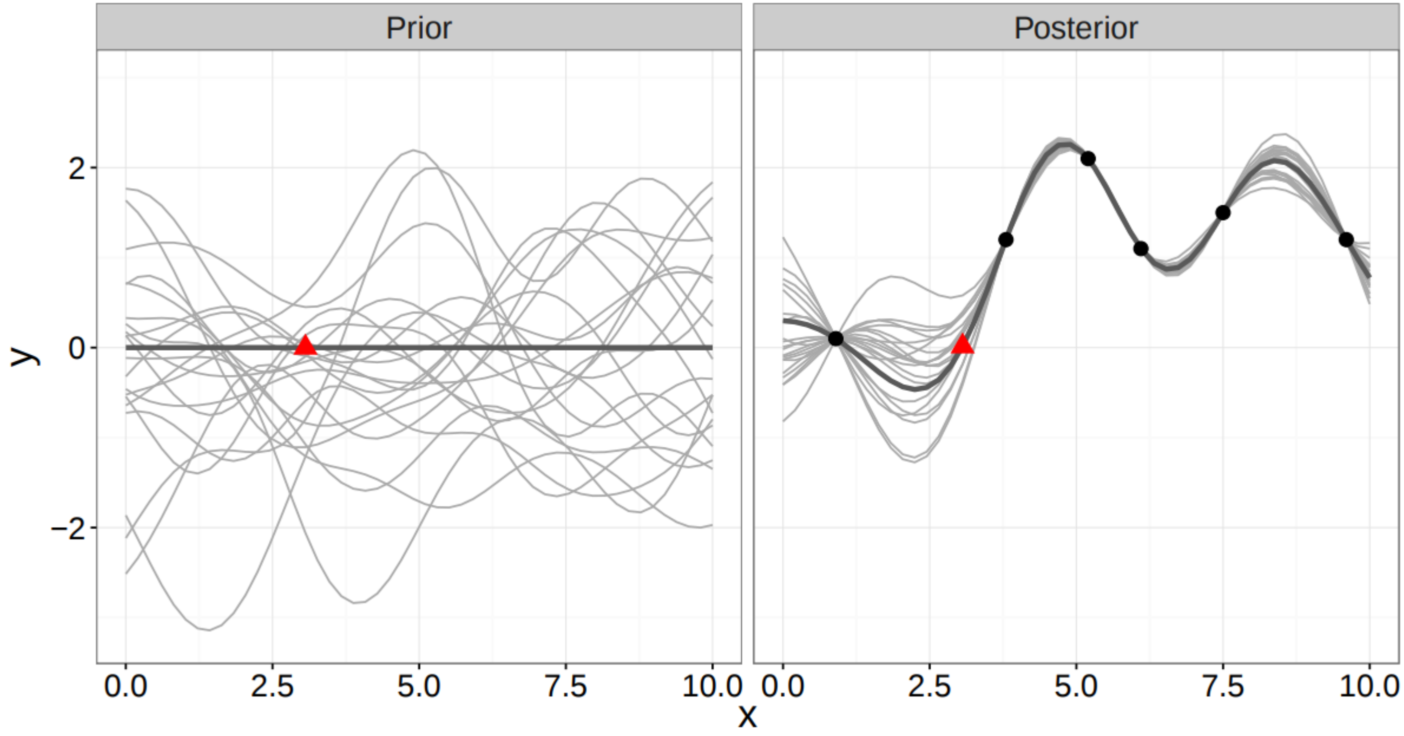
- Methods that require **weaker or less restrictive assumptions** are preferred.

    - **leads to estimating <u>flexible</u>* models e.g Gaussian Processes**

Gaussian Process

- Methods that require **weaker or less restrictive assumptions** are preferred.

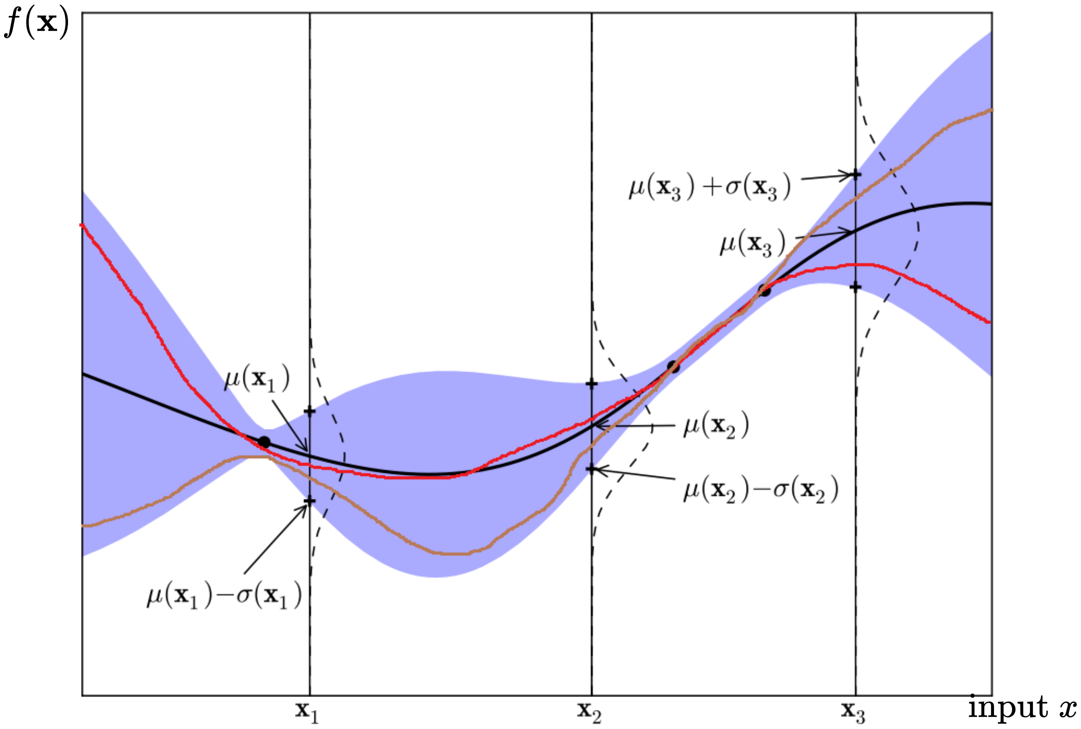
  - **leads to estimating <u>flexible</u>* models e.g Gaussian Processes**



**Ghahramani, 2016*

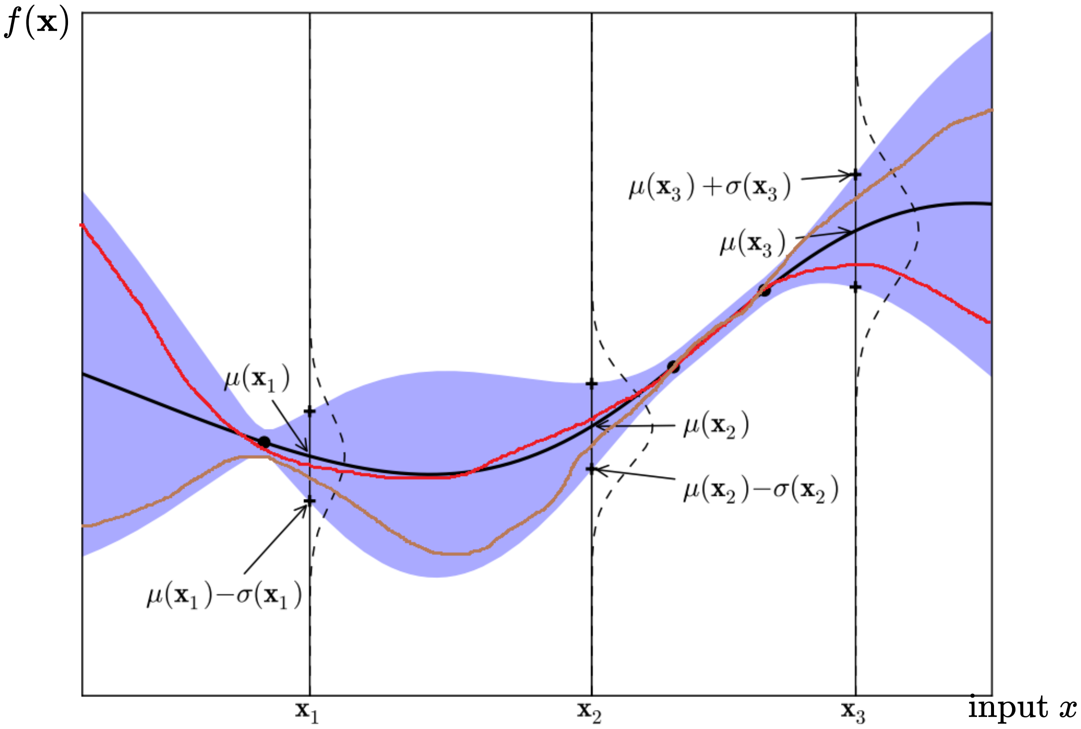
Define a prior over the function

- A distribution over functions in an infinite space is:
  - **a Gaussian process (GP)** (Rasmussen & Williams , 2006)



Define a prior over the function

- A distribution over functions in an infinite space is:
  - **a Gaussian process (GP)** (Rasmussen & Williams , 2006)



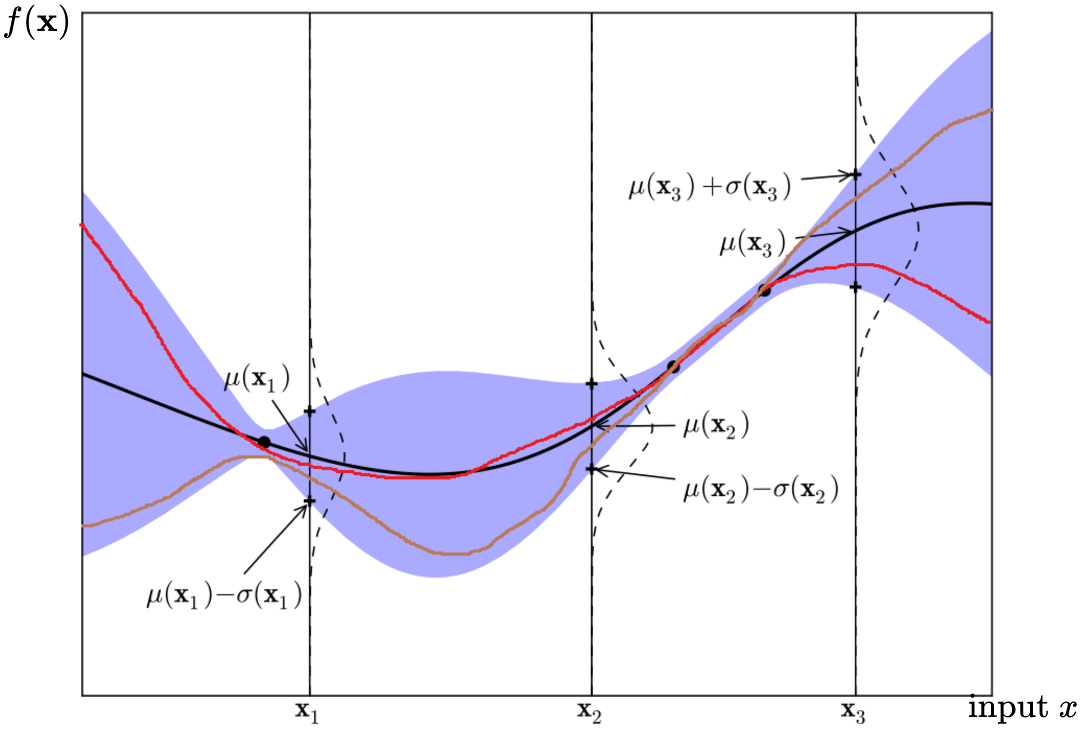
$$\mathbf{f} \sim \mathcal{GP}(\mathbf{m}_f, \mathbf{K}_f)$$

where

- $\mathbf{m}_f$ **= mean function**
- $\mathbf{K}_f$ **= covariance function (kernel)**

Define a prior over the function

- A distribution over functions in an infinite space is:
  - **a Gaussian process (GP)** (Rasmussen & Williams , 2006)



$$\mathbf{f} \sim \mathcal{GP}(\mathbf{m}_f, \mathbf{K}_f)$$

where

- $\mathbf{m}_f$ **= mean function**
- $\mathbf{K}_f$ **= covariance function (kernel)**

$$\begin{pmatrix} f(\mathbf{x}_1) \\ f(\mathbf{x}_2) \\ \vdots \\ f(\mathbf{x}_N) \end{pmatrix} \sim \mathcal{N}\left\{ \begin{pmatrix} m_f(\mathbf{x}_1) \\ m_f(\mathbf{x}_2) \\ \vdots \\ m_f(\mathbf{x}_N) \end{pmatrix}, \begin{pmatrix} k_f(\mathbf{x}_1, \mathbf{x}'_1) & k_f(\mathbf{x}_1, \mathbf{x}'_2) & \cdots & k_f(\mathbf{x}_1, \mathbf{x}'_N) \\ k_f(\mathbf{x}_2, \mathbf{x}'_1) & k_f(\mathbf{x}_2, \mathbf{x}'_2) & \cdots & k_f(\mathbf{x}_2, \mathbf{x}'_N) \\ \vdots & \vdots & \ddots & \vdots \\ k_f(\mathbf{x}_N, \mathbf{x}'_1) & k_f(\mathbf{x}_N, \mathbf{x}'_2) & \cdots & k_f(\mathbf{x}_N, \mathbf{x}'_N) \end{pmatrix} \right\}$$

See Appendix for mathematical details

Link

Probabilistic Programming

Probabilistic Programming (1/2)

- Probabilisic Programming (PP) Languages:
  - Software packages that take a model and then automatically generate inference routines (even source code!) e.g Pyro, Stan, Infer.Net, PyMC3, TensorFlow Probability, etc.

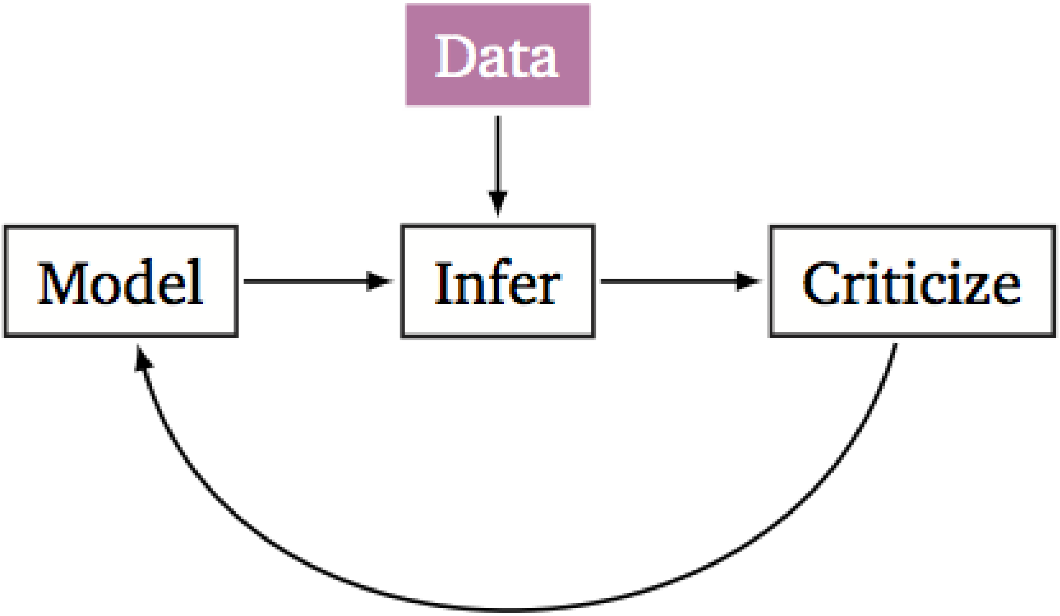
TensorFlow

PyMC3

infer.net

Probabilistic Programming (2/2)

- Steps in Probabilisic ML:
    - Build the model (Joint probability distribution of all the relevant variables)
    - Incorporate the observed data
    - Perform inference (to learn distributions of the latent variables)



Box's Loop (Image credit: http://dustintran.com/)

Gaussian Process in PyMC3

```
import pymc3 as pm

# Instantiate a model
with pm.Model() as latent_gp_model:

    # specify the priors
    length_scale = pm.Gamma("length_scale", alpha = 2, beta = 1)
    signal_variance = pm.HalfCauchy("signal_variance", beta = 5)
    noise_variance = pm.HalfCauchy("noise_variance", beta = 5)
    degrees_of_freedom = pm.Gamma("degrees_of_freedom", alpha = 2, beta = 0.1)

    # specify the kernel function
    cov = signal_variance**2 * pm.gp.cov.ExpQuad(1, length_scale)

    # specify the mean function
    mean_function = pm.gp.mean.Zero()

    # specify the gp
    gp = pm.gp.Latent(cov_func = cov)

    # specify the prior over the latent function
    f = gp.prior("f", X = X)

    # specify the likelihood
    obs = pm.StudentT("obs", mu = f, lam = 1/signal_variance, nu = degrees_of_freedom, observed = y)
```

Build a model

```
# Perform Inference
with latent_gp_model:
    posterior = pm.sample(draws = 100, njobs = 2)
```

Train a model

```
# extend the model by adding the GP conditional distribution so as to predict at test data
with latent_gp_model:
    f_pred = gp.conditional("f_pred", X_new)

# sample from the GP conditional posterior
with latent_gp_model:
    posterior_pred = pm.sample_ppc(posterior, vars = [f_pred], samples = 200)
```

Prediction

Scikit-learn

- Build + Train + predict + score + save + load

```
from sklearn.gaussian_process import GaussianProcessRegressor()

model = GaussianProcessRegressor()

model.fit(X_train, y_train)

model.predict(X_test, y_test)

model.score(X_test, y_test)

model.save('path/to/saved/model')
```

Few lines of code

scikit-learn.org

Pymc-learn

pymc-learn
latest

Search docs

- Introduction
- Getting started
- API Reference

Docs » Welcome to the documentation for `pymc-learn` !

Edit on GitHub

Welcome to the documentation for `pymc-learn`!

PyMC3

build unknown | coverage unknown | docs passing | license Apache 2 | launch binder

The purpose of this documentation is to describe the `pymc-learn` package: What it is, Why it is important, how to install and use it and how to contribute new ML algorithms.

Contents:

1. Github repo
2. What is pymc-learn?
3. Quick Install
4. Quick Start
5. Source Documentation
6. Contributing

pymc-learn.org

Pymc-learn

```
from pmlearn.gaussian_process import GaussianProcessRegressor()

# Instantiate a PyMC3 Gaussian process model
model = GaussianProcessRegressor()

# Fit using MCMC or Variational Inference
model.fit(X_train, y_train)

model.predict(X_test, y_test)

model.score(X_test, y_test)

model.save('path/to/saved/model')
```

Few lines of code

pymc-learn.org

PyMC3 vs Pymc-learn

```python
import pymc3 as pm

# Instantiate a model
with pm.Model() as latent_gp_model:

    # specify the priors
    length_scale = pm.Gamma("length_scale", alpha = 2, beta = 1)
    signal_variance = pm.HalfCauchy("signal_variance", beta = 5)
    noise_variance = pm.HalfCauchy("noise_variance", beta = 5)
    degrees_of_freedom = pm.Gamma("degrees_of_freedom", alpha = 2, beta = 0.1)

    # specify the kernel function
    cov = signal_variance**2 * pm.gp.cov.ExpQuad(1, length_scale)

    # specify the mean function
    mean_function = pm.gp.mean.Zero()

    # specify the gp
    gp = pm.gp.Latent(cov_func = cov)

    # specify the prior over the latent function
    f = gp.prior("f", X = X)

    # specify the likelihood
    obs = pm.StudentT("obs", mu = f, lam = 1/signal_variance, nu = degrees_of_freedom, observed = y)


# Perform Inference
with latent_gp_model:
    posterior = pm.sample(draws = 100, njobs = 2)

# extend the model by adding the GP conditional distribution so as to predict at test data
with latent_gp_model:
    f_pred = gp.conditional("f_pred", X_new)

# sample from the GP conditional posterior
with latent_gp_model:
    posterior_pred = pm.sample_ppc(posterior, vars = [f_pred], samples = 200)
```

Many lines of code

PyMC3

```python
from pmlearn.gaussian_process import GaussianProcessRegressor()

# Instantiate a PyMC3 Gaussian process model
model = GaussianProcessRegressor()

# Fit using MCMC or Variational Inference
model.fit(X_train, y_train)

model.predict(X_test, y_test)
```

Few lines of code

Pymc-learn

Demo

bit.ly/pymc-learn-dc

Resources to get started

- PyMC3 documents

- Winn, J., Bishop, C. M., Diethe, T. (2015). Model-Based Machine Learning. Microsoft Research Cambridge.

- R. McElreath (2012) Statistical Rethinking: A Bayesian Course with Examples in R and Stan (& PyMC3 & brms too)

- Probabilistic Programming and Bayesian Methods for Hackers: Fantastic book with many applied code examples.

Thank You!

Appendix

Start with an inflexible* model

- Consider for each data input, i, that:

y_i = output variable,

$\mathbf{x}_i$ = covariates with dimension D, e.g {income, employment, trip distance, etc}

$f(\mathbf{x}_i)$ = function that maps $\mathbf{x}_i$ to y_i,

ε_i = noise term,

$$y_i = f(\mathbf{x}_i) + \varepsilon_i, \quad \text{assume } \varepsilon \sim \mathcal{N}(0, \sigma_\varepsilon^2)$$

Ghahramani, 2016; Williams & Rasmussen, 2006

Start with an inflexible* model

- Consider for each data input, i, that:

y_i = output variable,

$\mathbf{x}_i$ = covariates with dimension D, e.g {income, employment, trip distance, etc}

$f(\mathbf{x}_i)$ = function that maps $\mathbf{x}_i$ to y_i,

ε_i = noise term,

$$y_i = f(\mathbf{x}_i) + \varepsilon_i, \quad \text{assume } \varepsilon \sim \mathcal{N}(0, \sigma_\varepsilon^2)$$

- Bayesian modeling involves:

$$p(\theta \mid \mathbf{y}, \mathbf{X}) = \frac{p(\mathbf{y}|\theta,\mathbf{X})\, p(\theta)}{p(\mathbf{y}|\mathbf{X})}$$

where

- θ = parameters e.g. coefficients

$p(\theta)$ = prior over the parameters

$p(\mathbf{y} \mid \theta, \mathbf{X})$ = likelihood of log activity duration, given the covariates & parameters

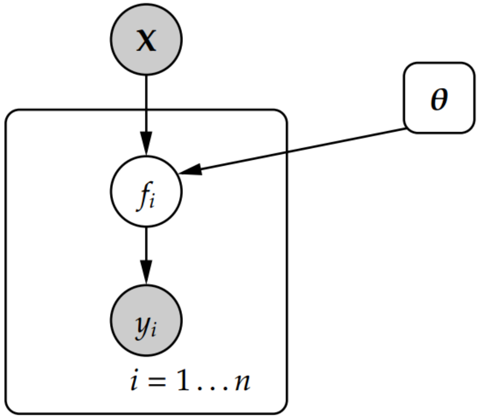
$p(\mathbf{y} \mid \mathbf{X})$ = data distribution to ensure normalization

$p(\theta \mid \mathbf{y}, \mathbf{X})$ = posterior over the parameters, given observed data

Ghahramani, 2016; Williams & Rasmussen, 2006

Extension to a flexible* model

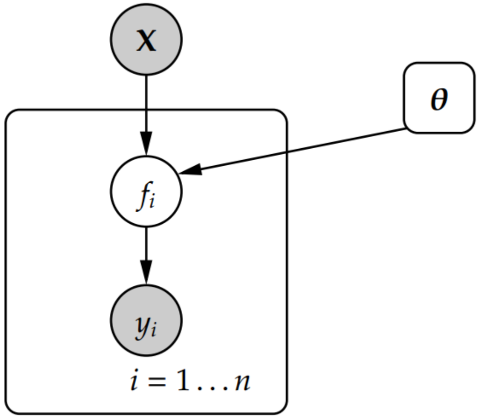
- Given that latent function f, is unknown:
  - **It (f) is considered the parameter of interes**



**Ghahramani, 2016; Williams & Rasmussen, 2006*

Extension to a flexible* model

- Given that latent function f, is unknown:
  - **It (f) is considered the parameter of interes**



$$p(\mathbf{f} \mid \mathbf{y}, \mathbf{X}) = \frac{p(\mathbf{y}|\mathbf{f},\mathbf{X})\, p(\mathbf{f})}{p(\mathbf{y}|\mathbf{X})}$$

$p(\mathbf{f})$ = **prior over the function**

$p(\mathbf{y} \mid \mathbf{f}, \mathbf{X})$ = likelihood of output variable, given the covariates & function

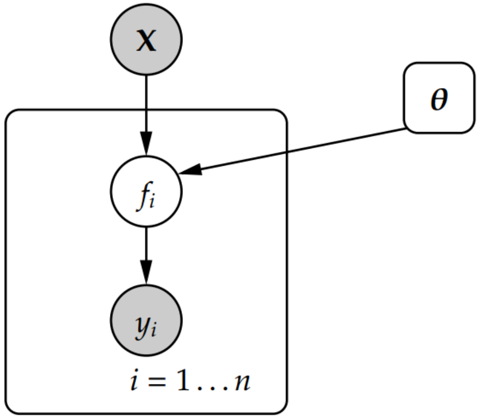
$p(\mathbf{y} \mid \mathbf{X})$ = data distribution to ensure normalization

$p(\mathbf{f} \mid \mathbf{y}, \mathbf{X})$ = posterior over the function, given observed data

*Ghahramani, 2016; Williams & Rasmussen, 2006

Extension to a flexible* model

- Given that latent function f, is unknown:
  - **It (f) is considered the parameter of interes**



$$p(\mathbf{f} \mid \mathbf{y}, \mathbf{X}) = \frac{p(\mathbf{y}|\mathbf{f},\mathbf{X})\, p(\mathbf{f})}{p(\mathbf{y}|\mathbf{X})}$$

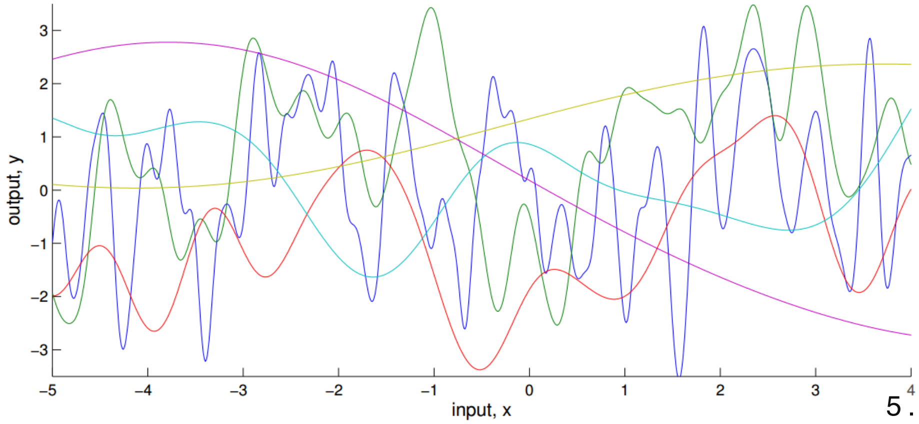
$p(\mathbf{f})$ = **prior over the function**

$p(\mathbf{y} \mid \mathbf{f}, \mathbf{X})$ = likelihood of output variable, given the covariates & function

$p(\mathbf{y} \mid \mathbf{X})$ = data distribution to ensure normalization

$p(\mathbf{f} \mid \mathbf{y}, \mathbf{X})$ = posterior over the function, given observed data

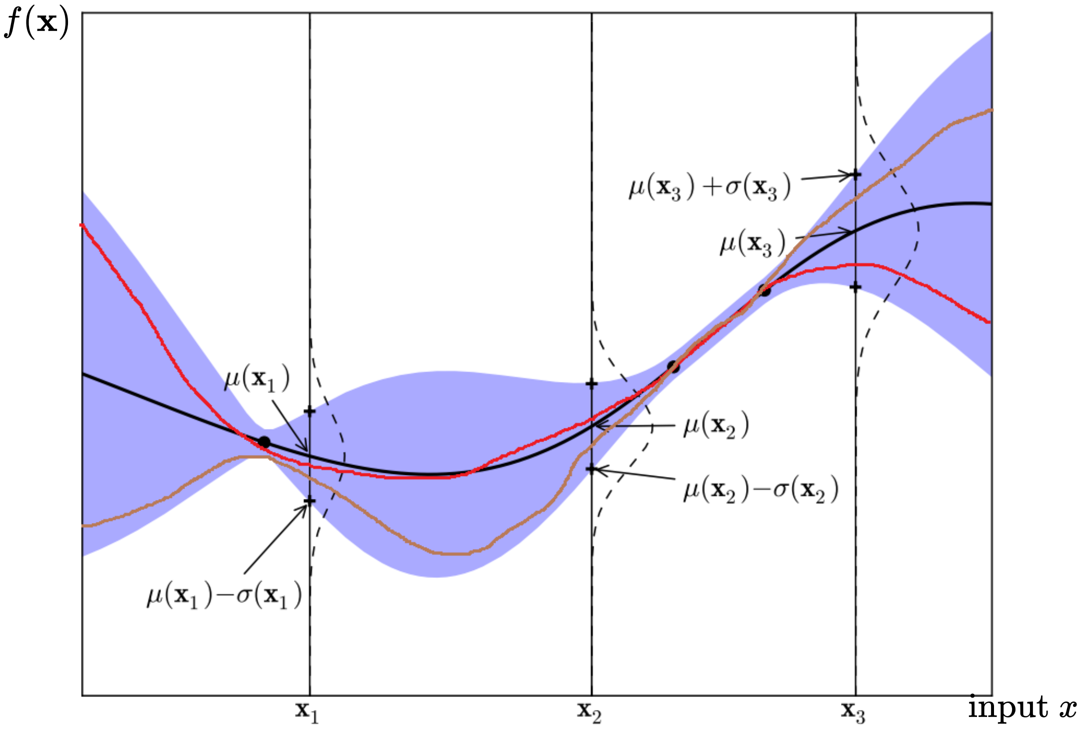
- Consider the prior over the function as:
  - **as any possible function in an infinite space**



Ghahramani, 2016; Williams & Rasmussen, 2006

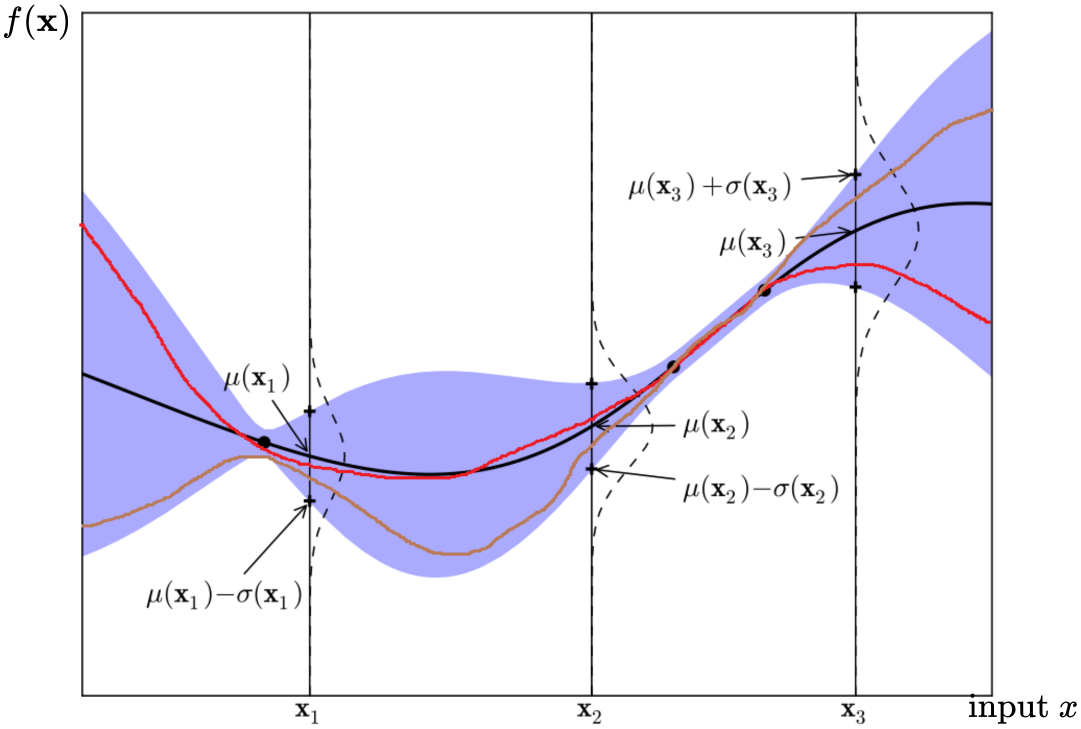
Define a prior over the function (1/2)

- A distribution over functions in an infinite space is:
  - **a Gaussian process (GP)** (Rasmussen & Williams , 2006)



Define a prior over the function (1/2)

- A distribution over functions in an infinite space is:
  - **a Gaussian process (GP)** (Rasmussen & Williams , 2006)



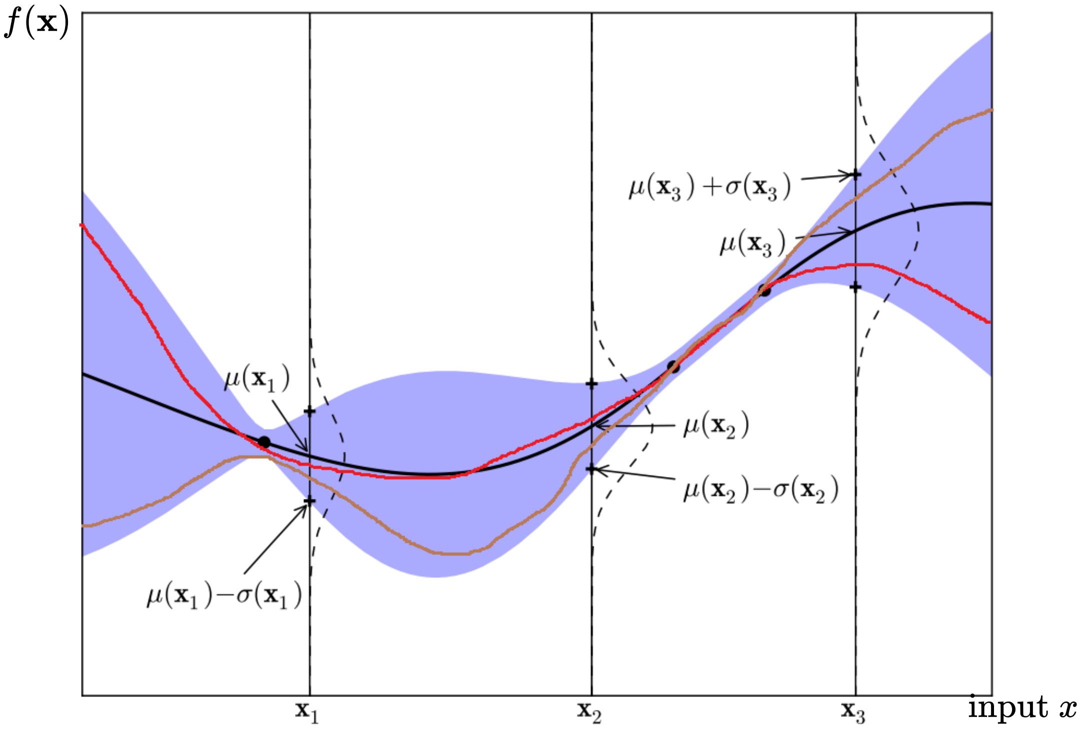
$$\mathbf{f} \sim \mathcal{GP}(\mathbf{m}_f, \mathbf{K}_f)$$

where

- **$\mathbf{m}_f$ = mean function**
- **$\mathbf{K}_f$ = covariance function (kernel)**

Define a prior over the function (1/2)

- A distribution over functions in an infinite space is:
  - **a Gaussian process (GP)** (Rasmussen & Williams , 2006)



$$\mathbf{f} \sim \mathcal{GP}(\mathbf{m}_f, \mathbf{K}_f)$$

where

- $\mathbf{m}_f$ **= mean function**
- $\mathbf{K}_f$ **= covariance function (kernel)**

$$\begin{pmatrix} f(\mathbf{x}_1) \\ f(\mathbf{x}_2) \\ \vdots \\ f(\mathbf{x}_N) \end{pmatrix} \sim \mathcal{N}\left\{ \begin{pmatrix} m_f(\mathbf{x}_1) \\ m_f(\mathbf{x}_2) \\ \vdots \\ m_f(\mathbf{x}_N) \end{pmatrix}, \begin{pmatrix} k_f(\mathbf{x}_1, \mathbf{x}'_1) & k_f(\mathbf{x}_1, \mathbf{x}'_2) & \cdots & k_f(\mathbf{x}_1, \mathbf{x}'_N) \\ k_f(\mathbf{x}_2, \mathbf{x}'_1) & k_f(\mathbf{x}_2, \mathbf{x}'_2) & \cdots & k_f(\mathbf{x}_2, \mathbf{x}'_N) \\ \vdots & \vdots & \ddots & \vdots \\ k_f(\mathbf{x}_N, \mathbf{x}'_1) & k_f(\mathbf{x}_N, \mathbf{x}'_2) & \cdots & k_f(\mathbf{x}_N, \mathbf{x}'_N) \end{pmatrix} \right\}$$

Define a prior over the function (2/2)

- Assumption:

We expect our functions to vary smoothly

Define a prior over the function (2/2)

- Assumption:

We expect our functions to vary smoothly

- The **Squared Exponential Automatic Relevance Determination** (SE-ARD) kernel (Neal, 2012).

$$k_f(\mathbf{x}_i, \mathbf{x}'_j \mid \theta) = \sigma_f^2 exp\left(-\frac{1}{2} \sum_{d=1}^{D} \frac{(x_{i,d} - x_{j,d})^2}{l_d^2} \right)$$

  - $\boldsymbol{\sigma_f^2}$ **= signal-variance parameter**
  - $\boldsymbol{l_d}$ **= length-scale parameter (**$\boldsymbol{l_d}$ **is positive; controls smoothness)**

Define a prior over the function (2/2)

- Assumption:

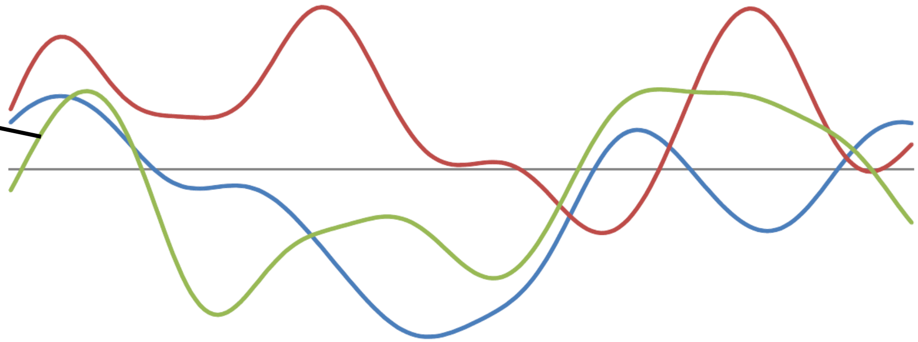
We expect our functions to vary smoothly

- The **Squared Exponential Automatic Relevance Determination** (SE-ARD) kernel (Neal, 2012).

$$k_f(\mathbf{x}_i, \mathbf{x}'_j \mid \theta) = \sigma_f^2 exp\left(- \frac{1}{2} \sum_{d=1}^{D} \frac{(x_{i,d} - x_{j,d})^2}{l_d^2} \right)$$

  - **σ_f^2 = signal-variance parameter**
  - **l_d = length-scale parameter (l_d is positive; controls smoothness)**

Very smooth
i.e continuous

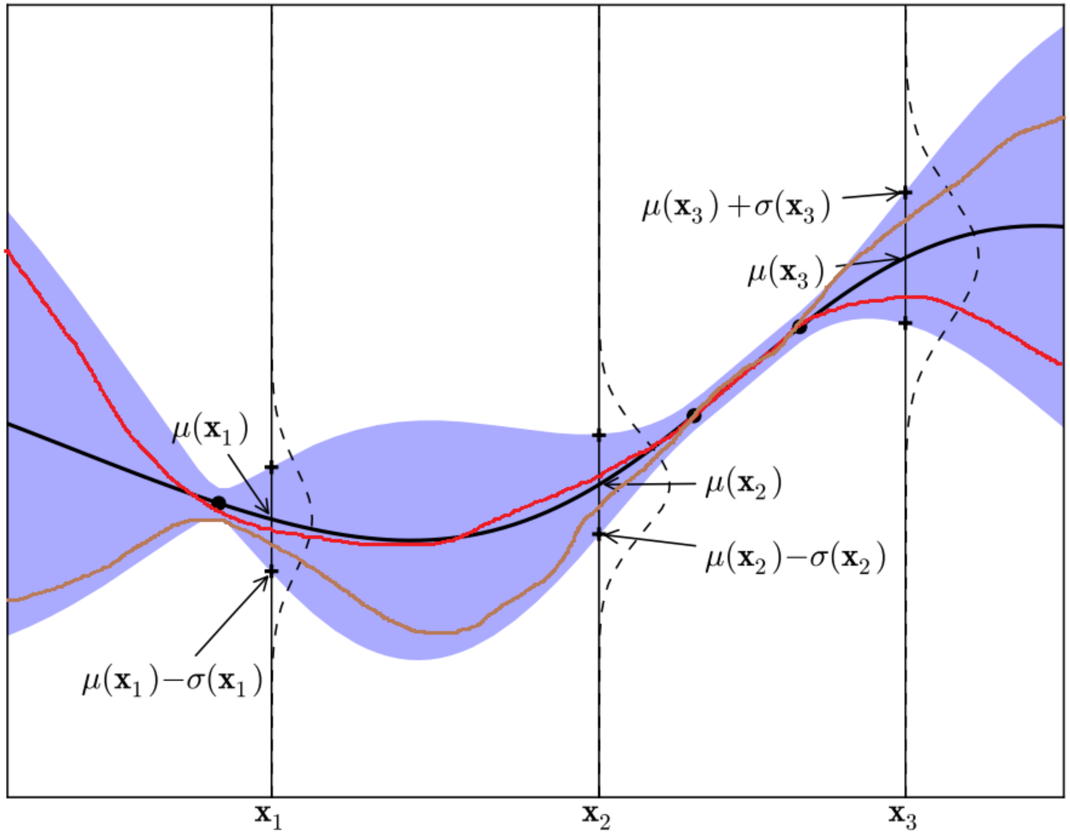


Efficient sampling

- Estimation will involve finding values for two sets hyperparameters:
    - σ_f^2 & σ_g^2 = signal-variance & noise variance
    - l_d^f & l_d^g = length-scales in f and g

Efficient sampling

- Estimation will involve finding values for two sets hyperparameters:
    - σ_f^2 & σ_g^2 = signal-variance & noise variance
    - l_d^f & l_d^g = length-scales in f and g



Efficient sampling

- Estimation will involve finding values for two sets hyperparameters:
  - σ_f^2 & σ_g^2 = signal-variance & noise variance
  - l_d^f & l_d^g = length-scales in f and g

- Recent sampling methods considered efficient in high dimensions will be used:
  - Hamiltonian Monte Carlo (HMC) i.e. No-U-turn Sampler (NUTS) (Hoffman & Gelman, 2014)
  - Automatic Differentiation Variational Inference (ADVI) (Kucukelbir et al, 2016)