CSC 411: Introduction to Machine Learning

CSC 411 Lecture 20: Gaussian Processes

Mengye Ren and Matthew MacKay

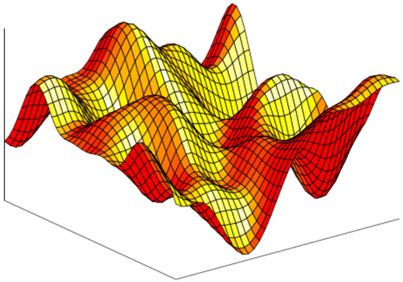
University of Toronto

Overview

- Last lecture: Bayesian linear regression, a parametric model
- This lecture: Gaussian processes
  - Derive as a generalization of Bayesian linear regression, with possibly infinitely many basis functions
  - Define a distribution directly over functions (i.e., a **stochastic process**)
  - Based on the Kernel Trick, one of the most important ideas in machine learning
  - Conceptually cleaner, since we can specify priors directly over functions. This lets us easily incorporate assumptions like smoothness, periodicity, etc., which are hard to encode as priors over regression weights.

Towards Gaussian Processes

- **Gaussian Processes** are distributions over functions.
- They're actually a simpler and more intuitive way to think about regression, once you're used to them.



— GPML

Towards Gaussian Processes

- A Bayesian linear regression model defines a distribution over functions:
$$f(\mathbf{x}) = \mathbf{w}^\top \psi(\mathbf{x})$$
Here, $\mathbf{w}$ is sampled from the prior $\mathcal{N}(\boldsymbol{\mu}_\mathbf{w}, \boldsymbol{\Sigma}_\mathbf{w})$.
- Let $\mathbf{f} = (f_1, \ldots, f_N)$ denote the vector of function values at $(\mathbf{x}_1, \ldots, \mathbf{x}_N)$.
- By the linear transformation rules for Gaussian random variables, the distribution of $\mathbf{f}$ is a Gaussian with
$$\mathbb{E}[f_i] = \boldsymbol{\mu}_\mathbf{w}^\top \psi(\mathbf{x})$$
$$\mathrm{Cov}(f_i, f_j) = \psi(\mathbf{x}_i)^\top \boldsymbol{\Sigma}_\mathbf{w} \psi(\mathbf{x}_j)$$
- In vectorized form, $\mathbf{f} \sim \mathcal{N}(\boldsymbol{\mu}_\mathbf{f}, \boldsymbol{\Sigma}_\mathbf{f})$ with
$$\boldsymbol{\mu}_\mathbf{f} = \mathbb{E}[\mathbf{f}] = \boldsymbol{\Psi}\boldsymbol{\mu}_\mathbf{w}$$
$$\boldsymbol{\Sigma}_\mathbf{f} = \mathrm{Cov}(\mathbf{f}) = \boldsymbol{\Psi}\boldsymbol{\Sigma}_\mathbf{w}\boldsymbol{\Psi}^\top$$

Towards Gaussian Processes

- Recall that in Bayesian linear regression, we assume noisy Gaussian observations of the underlying function.

$$y_i \sim \mathcal{N}(f_i, \sigma^2) = \mathcal{N}(\mathbf{w}^\top \psi(\mathbf{x}_i), \sigma^2).$$

- The observations $\mathbf{y}$ are jointly Gaussian, just like $\mathbf{f}$.

$$\mathbb{E}[y_i] = \mathbb{E}[f(\mathbf{x}_i)]$$

$$\mathrm{Cov}(y_i, y_j) = \begin{cases} \mathrm{Var}(f(\mathbf{x}_i)) + \sigma^2 & \text{if } i = j \\ \mathrm{Cov}(f(\mathbf{x}_i), f(\mathbf{x}_j)) & \text{if } i \neq j \end{cases}$$

- In vectorized form, $\mathbf{y} \sim \mathcal{N}(\boldsymbol{\mu}_\mathbf{y}, \boldsymbol{\Sigma}_\mathbf{y})$, with

$$\boldsymbol{\mu}_\mathbf{y} = \boldsymbol{\mu}_\mathbf{f}$$
$$\boldsymbol{\Sigma}_\mathbf{y} = \boldsymbol{\Sigma}_\mathbf{f} + \sigma^2 \mathbf{I}$$

Towards Gaussian Processes

- Bayesian linear regression is just computing the conditional distribution in a multivariate Gaussian
- Let $\mathbf{y}$ and $\mathbf{y}'$ denote the observables at the training and test data.
- They are jointly Gaussian:

$$\begin{pmatrix} \mathbf{y} \\ \mathbf{y}' \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \boldsymbol{\mu}_{\mathbf{y}} \\ \boldsymbol{\mu}_{\mathbf{y}'} \end{pmatrix}, \begin{pmatrix} \boldsymbol{\Sigma}_{\mathbf{yy}} & \boldsymbol{\Sigma}_{\mathbf{yy}'} \\ \boldsymbol{\Sigma}_{\mathbf{y}'\mathbf{y}} & \boldsymbol{\Sigma}_{\mathbf{y}'\mathbf{y}'} \end{pmatrix} \right).$$

- The predictive distribution is a special case of the conditioning formula for a multivariate Gaussian:

$$\begin{aligned} \mathbf{y}' \mid \mathbf{y} &\sim \mathcal{N}(\boldsymbol{\mu}_{\mathbf{y}'|\mathbf{y}}, \boldsymbol{\Sigma}_{\mathbf{y}'|\mathbf{y}}) \\ \boldsymbol{\mu}_{\mathbf{y}'|\mathbf{y}} &= \boldsymbol{\mu}_{\mathbf{y}'} + \boldsymbol{\Sigma}_{\mathbf{y}'\mathbf{y}} \boldsymbol{\Sigma}_{\mathbf{yy}}^{-1} (\mathbf{y} - \boldsymbol{\mu}_{\mathbf{y}}) \\ \boldsymbol{\Sigma}_{\mathbf{y}'|\mathbf{y}} &= \boldsymbol{\Sigma}_{\mathbf{y}'\mathbf{y}'} - \boldsymbol{\Sigma}_{\mathbf{y}'\mathbf{y}} \boldsymbol{\Sigma}_{\mathbf{yy}}^{-1} \boldsymbol{\Sigma}_{\mathbf{yy}'} \end{aligned}$$

- We're implicitly marginalizing out $\mathbf{w}$

Towards Gaussian Processes

- The marginal likelihood is just the PDF of a multivariate Gaussian:

$$\begin{aligned} p(\mathbf{y} \mid \mathbf{X}) &= \mathcal{N}(\mathbf{y}; \boldsymbol{\mu}_{\mathbf{y}}, \boldsymbol{\Sigma}_{\mathbf{y}}) \\ &= \frac{1}{(2\pi)^{d/2} |\boldsymbol{\Sigma}_{\mathbf{y}}|^{1/2}} \exp\left(-\frac{1}{2} (\mathbf{y} - \boldsymbol{\mu}_{\mathbf{y}})^\top \boldsymbol{\Sigma}_{\mathbf{y}}^{-1} (\mathbf{y} - \boldsymbol{\mu}_{\mathbf{y}}) \right) \end{aligned}$$

Towards Gaussian Processes

- To summarize:

$$
\begin{aligned}
\boldsymbol{\mu}_{\mathbf{f}} &= \boldsymbol{\Psi}\boldsymbol{\mu}_{\mathbf{w}} \\
\boldsymbol{\Sigma}_{\mathbf{f}} &= \boldsymbol{\Psi}\boldsymbol{\Sigma}_{\mathbf{w}}\boldsymbol{\Psi}^{\top} \\
\boldsymbol{\mu}_{\mathbf{y}} &= \boldsymbol{\mu}_{\mathbf{f}} \\
\boldsymbol{\Sigma}_{\mathbf{y}} &= \boldsymbol{\Sigma}_{\mathbf{f}} + \sigma^2\mathbf{I} \\
\boldsymbol{\mu}_{\mathbf{y}'|\mathbf{y}} &= \boldsymbol{\mu}_{\mathbf{y}'} + \boldsymbol{\Sigma}_{\mathbf{y}'\mathbf{y}}\boldsymbol{\Sigma}_{\mathbf{yy}}^{-1}(\mathbf{y} - \boldsymbol{\mu}_{\mathbf{y}}) \\
\boldsymbol{\Sigma}_{\mathbf{y}'|\mathbf{y}} &= \boldsymbol{\Sigma}_{\mathbf{y}'\mathbf{y}'} - \boldsymbol{\Sigma}_{\mathbf{y}'\mathbf{y}}\boldsymbol{\Sigma}_{\mathbf{yy}}^{-1}\boldsymbol{\Sigma}_{\mathbf{yy}'} \\
p(\mathbf{y} \,|\, \mathbf{X}) &= \mathcal{N}(\mathbf{y}; \boldsymbol{\mu}_{\mathbf{y}}, \boldsymbol{\Sigma}_{\mathbf{y}})
\end{aligned}
$$

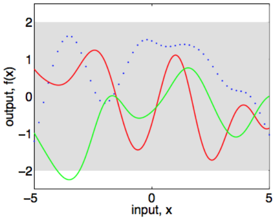
- After defining $\boldsymbol{\mu}_{\mathbf{f}}$ and $\boldsymbol{\Sigma}_{\mathbf{f}}$, we can forget about $\mathbf{w}$
- What if we just let $\boldsymbol{\mu}_{\mathbf{f}}$ and $\boldsymbol{\Sigma}_{\mathbf{f}}$ be other forms?

Gaussian Processes

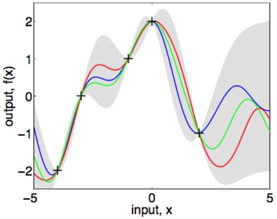
- We need to specify
  - a mean function $\mathbb{E}[f(\mathbf{x}_i)] = \mu(\mathbf{x}_i)$
  - a covariance function called a **kernel function**: $\mathrm{Cov}(f(\mathbf{x}_i), f(\mathbf{x}_j)) = k(\mathbf{x}_i, \mathbf{x}_j)$
- Let $\mathbf{K}_\mathbf{X}$ denote the kernel matrix for points $\mathbf{X}$. This is a matrix whose (i, j) entry is $k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$, and is called the **Gram matrix**.
- We require that $\mathbf{K}_\mathbf{X}$ be positive semidefinite for *any* $\mathbf{X}$. Other than that, μ and k can be arbitrary.

Gaussian Processes

- We've just defined a distribution over *function values* at an arbitrary finite set of points.
- This can be extended to a distribution over *functions* using Kolmogorov Extension Theorem. This distribution over functions is called a **Gaussian process (GP)**.
- But distributions over functions are conceptually cleaner.



(a), prior



(b), posterior

- How are these plots were generated?

Kernel Trick

- This is an instance of a more general trick called the **Kernel Trick**.
- Many algorithms (e.g. linear regression, logistic regression, SVMs) can be written in terms of dot products between feature vectors, $\boldsymbol{\psi}(\mathbf{x})^\top \boldsymbol{\psi}(\mathbf{x}')$.
- A **kernel** implements an inner product between feature vectors, typically implicitly, and often much more efficiently than the explicit dot product.
- For instance, the following feature vector is quadratic in size:

$$\phi(\mathbf{x}) = (1, \sqrt{2}x_1, ..., \sqrt{2}x_d, \sqrt{2}x_1x_2, \sqrt{2}x_1x_3, ...\sqrt{2}x_{d-1}x_d, x_1^2, ..., x_d^2)$$

- But the **quadratic kernel** can compute the inner product in linear time:

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \phi(\mathbf{x}') = 1 + \sum_{i=1}^{d} 2x_i x_i' + \sum_{i,j=1}^{d} x_i x_j x_i' x_j' = (1 + \mathbf{x}^\top \mathbf{x}')^2$$

SVM & Kernels

- Convert the constrained minimization to an unconstrained optimization problem: represent constraints as penalty terms:

$$\min_{\boldsymbol{w},b} \frac{1}{2}||\boldsymbol{w}||^2 + \text{penalty\_term}$$

- For data $\{(\phi(\boldsymbol{x}^{(i)}), t^{(i)})\}_{i=1}^{N}$, use the following penalty

$$\max_{\alpha_i \geq 0} \quad \alpha_i[1 - (\boldsymbol{w}^T\phi(\boldsymbol{x}^{(i)}) + b)t^{(i)}] = \begin{cases} 0 & \text{if} \quad (\boldsymbol{w}^T\phi(\boldsymbol{x}^{(i)}) + b)t^{(i)} \geq 1 \\ \infty & \text{otherwise} \end{cases}$$

- Rewrite the minimization problem

$$\min_{\boldsymbol{w},b} \left\{ \frac{1}{2}||\boldsymbol{w}||^2 + \sum_{i=1}^{N} \max_{\alpha_i \geq 0} \alpha_i[1 - (\boldsymbol{w}^T\phi(\boldsymbol{x}^{(i)}) + b)t^{(i)}] \right\}$$

where α_i are the **Lagrange multipliers**

$$= \min_{\boldsymbol{w},b} \max_{\alpha_i \geq 0} \left\{ \frac{1}{2}||\boldsymbol{w}||^2 + \sum_{i=1}^{N} \alpha_i[1 - (\boldsymbol{w}^T\phi(\boldsymbol{x}^{(i)}) + b)t^{(i)}] \right\}$$

SVM & Kernels

- Let:

$$J(\boldsymbol{w}, b; \alpha) = \frac{1}{2}||\boldsymbol{w}||^2 + \sum_{i=1}^{N} \alpha_i [1 - (\boldsymbol{w}^T \phi(\boldsymbol{x}^{(i)}) + b) t^{(i)}]$$

- Swap the "max" and "min": This is a lower bound

$$\max_{\alpha_i \geq 0} \min_{\boldsymbol{w}, b} J(\boldsymbol{w}, b; \alpha) \leq \min_{\boldsymbol{w}, b} \max_{\alpha_i \geq 0} J(\boldsymbol{w}, b; \alpha)$$

- Equality holds in certain conditions

SVM & Kernels

- Solving:

$$\max_{\alpha_i \geq 0} \min_{\mathbf{w},b} J(\mathbf{w}, b; \alpha) = \max_{\alpha_i \geq 0} \min_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||^2 + \sum_{i=1}^{N} \alpha_i [1 - (\mathbf{w}^T \mathbf{x}^{(i)} + b) t^{(i)}]$$

- First minimize $J()$ w.r.t. $\mathbf{w}, b$ for fixed Lagrange multipliers:

$$\begin{aligned} \frac{\partial J(\mathbf{w}, b; \alpha)}{\partial \mathbf{w}} &= \mathbf{w} - \sum_{i=1}^{N} \alpha_i \phi(\mathbf{x}^{(i)}) t^{(i)} = 0 \\ \frac{\partial J(\mathbf{w}, b; \alpha)}{\partial b} &= -\sum_{i=1}^{N} \alpha_i t^{(i)} = 0 \end{aligned}$$

- We obtain $\mathbf{w} = \sum_{i=1}^{N} \alpha_i t^{(i)} \phi(\mathbf{x}^{(i)})$
- Then substitute back to get final optimization:

$$L = \max_{\alpha_i \geq 0} \left\{ \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} t^{(i)} t^{(j)} \alpha_i \alpha_j \underbrace{(\phi(\mathbf{x}^{(i)})^T \phi(\mathbf{x}^{(j)}))}_{K(x^{(i)}, x^{(j)})} \right\}$$

Kernel Trick

- Many algorithms can be **kernelized**, i.e. written in terms of kernels, rather than explicit feature representations.
- We rarely think about the underlying feature space explicitly. Instead, we build kernels directly.
- Useful composition rules for kernels:
  - A constant function $k(\mathbf{x}, \mathbf{x}') = \alpha$ is a kernel.
  - If k_1 and k_2 are kernels and $a, b \geq 0$, then $ak_1 + bk_2$ is a kernel.
  - If k_1 and k_2 are kernels, then the product $k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}')$ is a kernel.
- Before neural nets took over, kernel SVMs were probably the best-performing general-purpose classification algorithm.

Kernel Trick: Computational Cost

- The kernel trick lets us implicitly use very high-dimensional (even infinite-dimensional) feature spaces, but this comes at a cost.
- **Bayesian linear regression:**

$$\boldsymbol{\mu} = \sigma^{-2}\boldsymbol{\Sigma}\boldsymbol{\Psi}^\top \mathbf{t}$$
$$\boldsymbol{\Sigma}^{-1} = \sigma^{-2}\boldsymbol{\Psi}^\top\boldsymbol{\Psi} + \mathbf{S}^{-1}$$

  - Need to compute the inverse of a $D \times D$ matrix, which is an $\mathcal{O}(D^3)$ operation. (D is the number of features.)
- **GP regression:**

$$\boldsymbol{\mu}_{\mathbf{y}'|\mathbf{y}} = \boldsymbol{\mu}_{\mathbf{y}'} + \boldsymbol{\Sigma}_{\mathbf{y}'\mathbf{y}}\boldsymbol{\Sigma}_{\mathbf{y}\mathbf{y}}^{-1}(\mathbf{y} - \boldsymbol{\mu}_{\mathbf{y}})$$
$$\boldsymbol{\Sigma}_{\mathbf{y}'|\mathbf{y}} = \boldsymbol{\Sigma}_{\mathbf{y}'\mathbf{y}'} - \boldsymbol{\Sigma}_{\mathbf{y}'\mathbf{y}}\boldsymbol{\Sigma}_{\mathbf{y}\mathbf{y}}^{-1}\boldsymbol{\Sigma}_{\mathbf{y}\mathbf{y}'}$$

  - Need to invert an $N \times N$ matrix! (N is the number of training examples.)

Kernel Trick: Computational Cost

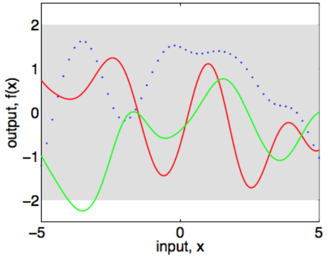
- This $\mathcal{O}(N^3)$ cost is typical of kernel methods. Most exact kernel methods don't scale to more than a few thousand data points.
- Kernel SVMs can be scaled further, since you can show you only need to consider the kernel over the support vectors, not the entire training set.
- Scaling GP methods to large datasets is an active (and fascinating) research area.

GP Kernels

- One way to define a kernel function is to give a set of basis functions and put a Gaussian prior on $\mathbf{w}$.
- But we have lots of other options. Here's a useful one, called the **squared-exp**, or **Gaussian**, or **radial basis function (RBF)** kernel:

$$k_{\mathrm{SE}}(\mathbf{x}_i, \mathbf{x}_j) = \sigma^2 \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\ell^2}\right)$$

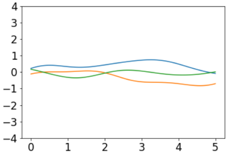
- More accurately, this is a **kernel family** with **hyperparameters** σ and ℓ.
- It gives a distribution over smooth functions:



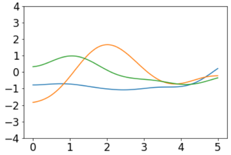
GP Kernels

$$k_{\mathrm{SE}}(x_i, x_j) = \sigma^2 \exp\left(-\frac{(x_i - x_j)^2}{2\ell^2}\right)$$

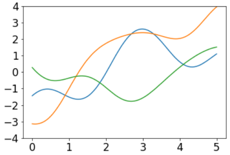
- The hyperparameters determine key properties of the function.
- Varying the **output variance** σ^2:



$\sigma^2 = 0.3$

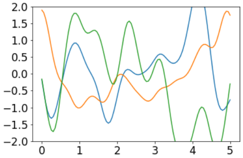


$\sigma^2 = 1$

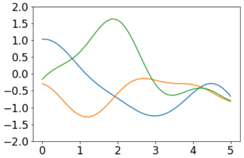


$\sigma^2 = 3$

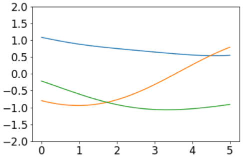
- Varying the **lengthscale** ℓ:



$\ell = 0.3$



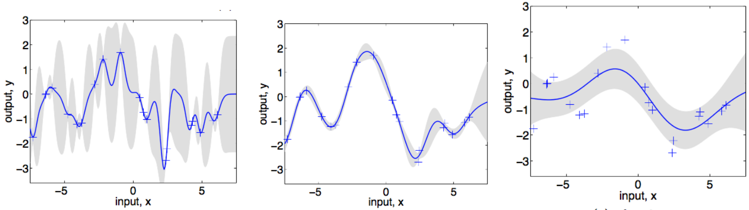
$\ell = 1$



$\ell = 3$

GP Kernels

- The choice of hyperparameters heavily influences the predictions:



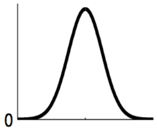
(b), $\ell = 0.3$ (a), $\ell = 1$ (c), $\ell = 3$

- In practice, it's very important to tune the hyperparameters (e.g. by maximizing the marginal likelihood).

GP Kernels

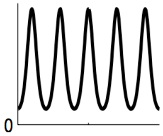
$$k_{\mathrm{SE}}(x_i, x_j) = \sigma^2 \exp\left(-\frac{(x_i - x_j)^2}{2\ell^2}\right)$$

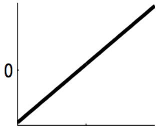
- The squared-exp kernel is **stationary** because it only depends on $x_i - x_j$. Most kernels we use in practice are stationary.
- We can visualize the function $k(0, x)$:

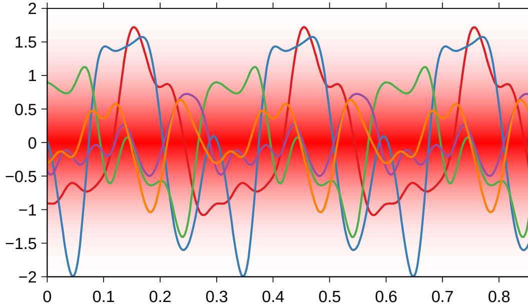


GP Kernels (optional)

- The periodic kernel encodes for a probability distribution over periodic functions
- The linear kernel results in a probability distribution over linear functions





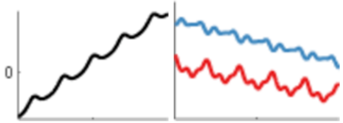


GP Kernels (optional)

- We get exponentially more flexibility by combining kernels.
- The sum of two kernels is a kernel.
  - This is because valid covariance matrices (i.e. PSD matrices) are closed under addition.
- The sum of two kernels corresponds to the sum of functions.

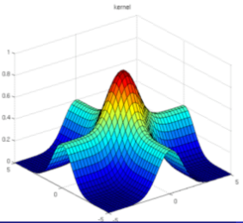
Linear + Periodic

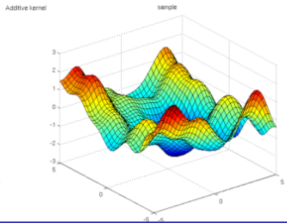
e.g. seasonal pattern w/ trend



Additive kernel

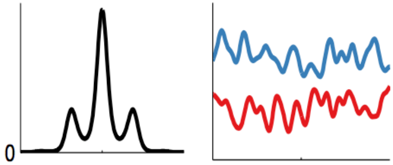
$$k(x, y, x', y') = k_1(x, x') + k_2(y, y')$$





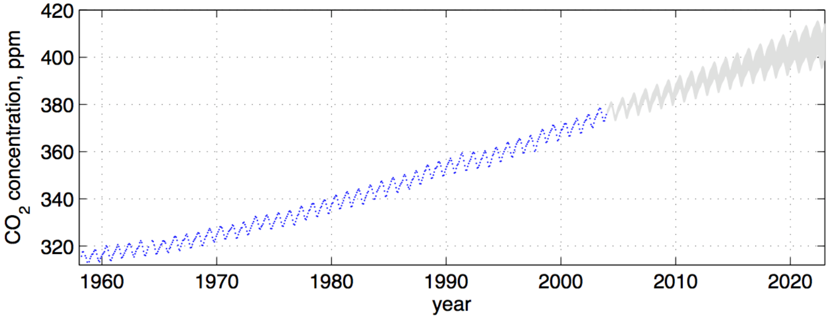
GP Kernels (optional)

- A kernel is like a similarity function on the input space. The sum of two kernels is like the OR of their similarity.
- Amazingly, the product of two kernels is a kernel. (Follows from the Schur Product Theorem.)
- The product of two kernels is like the AND of their similarity functions.
- Example: the product of a squared-exp kernel (spatial similarity) and a periodic kernel (similar location within cycle) gives a locally periodic function.



GP Kernels (optional)

- Modeling CO2 concentrations:
  trend + (changing) seasonal pattern + short-term variability + noise
- Encoding the structure allows sensible extrapolation.



Summary

- Bayesian linear regression lets us determine uncertainty in our predictions.
- Bayesian Occam's Razor is a sophisticated way of penalizing the complexity of a distribution over functions.
- Gaussian processes are an elegant framework for doing Bayesian inference directly over functions.
- The choice of kernels gives us much more control over what sort of functions our prior would allow or favor.