Probabilistic Machine Learning
Lecture 08
Gaussian Proceses

Philipp Hennig

15 May 2023

Eberhard Karls Universität Tübingen

Faculty of Science
Department of Computer Science
Chair for the Methods of Machine Learning

Summary:

- Gaussian distributions can be used to learn functions
- Analytical inference is possible using linear models

$$f(x) = \phi(x)^\intercal \mathbf{w} = \phi_x^\intercal \mathbf{w}$$

- The choice of features $\phi : \mathbb{X} \rightharpoonup \mathbb{R}$ is essentially unconstrained

$$f(x) := \phi_x^\intercal \boldsymbol{w} \qquad p(w) = \mathcal{N}(w; \mu, \Sigma) \qquad p(\boldsymbol{y} \mid f) = \mathcal{N}(\boldsymbol{y}; f_X, \sigma^2 I)$$

$$\begin{aligned}
p(w \mid \boldsymbol{y}, \phi_X) &= \mathcal{N}\left(w; \mu + \Sigma\phi_X(\phi_X^\intercal \Sigma \phi_X + \sigma^2 I)^{-1}(\boldsymbol{y} - \phi_X^\intercal \mu), \Sigma - \Sigma\phi_X(\phi_X^\intercal \Sigma \phi_X + \sigma^2 I)^{-1}\phi_X^\intercal \Sigma\right) \\
&= \mathcal{N}(w; \mu_y, \Sigma_y) \qquad \text{where} \\
\underbrace{\mu_y}_{\mathbb{E}(\boldsymbol{w}|\boldsymbol{y})} &:= \underbrace{\mu}_{\mathbb{E}(\boldsymbol{w})} + \underbrace{\Sigma\phi_X}_{\operatorname{cov}(\boldsymbol{w},\boldsymbol{y})} \underbrace{(\phi_X^\intercal \Sigma \phi_X + \sigma^2)^{-1}}_{\operatorname{cov}(\boldsymbol{y},\boldsymbol{y})^{-1}} \underbrace{(\boldsymbol{y} - \phi_X^\intercal \mu)}_{\boldsymbol{y} - \mathbb{E}(\boldsymbol{y})} \qquad \text{and} \\
\underbrace{\Sigma_y}_{\operatorname{cov}(\boldsymbol{w},\boldsymbol{w}|\boldsymbol{y})} &:= \underbrace{\Sigma}_{\operatorname{cov}(\boldsymbol{w},\boldsymbol{w})} - \underbrace{\Sigma\phi_X}_{\operatorname{cov}(\boldsymbol{w},\boldsymbol{y})} \underbrace{(\phi_X^\intercal \Sigma \phi_X + \sigma^2)^{-1}}_{\operatorname{cov}(\boldsymbol{y},\boldsymbol{y})^{-1}} \underbrace{\phi_X^\intercal \Sigma}_{\operatorname{cov}(\boldsymbol{y},\boldsymbol{w})} \\
p(f_x \mid \boldsymbol{y}, \phi_X) &= \mathcal{N}\left(f_x; \phi_x^\intercal \mu_y, \phi_x^\intercal \Sigma_y \phi_x\right) \\
&= \mathcal{N}\Big(f_x; \phi_x^\intercal \mu + \phi_x^\intercal \Sigma\phi_X(\phi_X^\intercal \Sigma \phi_X + \sigma^2 I)^{-1}(y - \phi_X^\intercal \mu), \\
&\qquad \phi_x^\intercal \left(\Sigma -^\intercal \Sigma\phi_X(\phi_X^\intercal \Sigma \phi_X + \sigma^2 I)^{-1}\phi_X^\intercal \Sigma\right)\phi_x\Big)
\end{aligned}$$

Some Things to Note

Probabilistic Parametric Regression

$$f(x) := \phi_X^\intercal \boldsymbol{w} \qquad p(w) = \mathcal{N}(w; \mu, \Sigma) \qquad p(\boldsymbol{y} \mid f) = \mathcal{N}(\boldsymbol{y}; f_X, \sigma^2 I)$$

$$\begin{aligned}
p(w \mid \boldsymbol{y}, \phi_X) &= \mathcal{N}\left(w; \mu + \Sigma\phi_X(\phi_X^\intercal \Sigma \phi_X + \sigma^2 I)^{-1}(\boldsymbol{y} - \phi_X^\intercal \mu), \Sigma - \Sigma\phi_X(\phi_X^\intercal \Sigma \phi_X + \sigma^2 I)^{-1}\phi_X^\intercal \Sigma\right) \\
&= \mathcal{N}(w; \mu_y, \Sigma_y) \qquad \text{where} \\
\mu_y &:= \mu + \underbrace{\Sigma\phi_X(\phi_X^\intercal \Sigma \phi_X + \sigma^2)^{-1}}_{\nabla_y \mu_y}(\boldsymbol{y} - \phi_X^\intercal \mu) \quad \text{and} \\
\Sigma_y &:= \Sigma - \Sigma\phi_X(\phi_X^\intercal \Sigma \phi_X + \sigma^2)^{-1}\phi_X^\intercal \Sigma. \qquad \text{Thus,} \\
&= \Sigma - \Sigma\phi_X(\nabla_y \mu_y)^\intercal
\end{aligned}$$

Computing the posterior covariance is **cheap**! It measures the *remaining capacity* in the model after seeing the data. **If you can auto-diff, you can be uncertain!**

Shallow Learning

Gaussian linear regressors are **single-layer neural networks** with quadratic loss

$$p(\mathbf{w} \mid \mathbf{y}, X) = \frac{1}{p(\mathbf{y})}\mathcal{N}(\mathbf{y} \mid \phi_X^\intercal \mathbf{w}, \sigma I)\mathcal{N}(\mathbf{w}; \mu, \Sigma)$$

$$-\log p(\mathbf{w} \mid \mathbf{y}, X) = -\log \mathcal{N}(\mathbf{y} \mid \phi_X^\intercal \mathbf{w}, \sigma I) - \log \mathcal{N}(\mathbf{w}; \mu, \Sigma) + \text{const.}$$

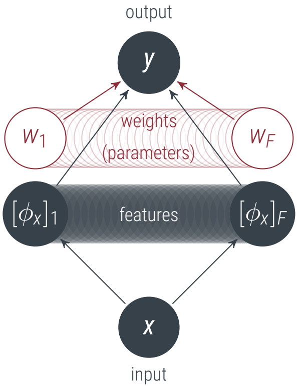
$$\log p(\mathbf{w} \mid \mathbf{y}, X) \triangleq \frac{1}{2\sigma^2}\sum_i \|y_i - \phi(x_i)^\intercal \mathbf{w}\|^2 + \frac{1}{2}(\mathbf{w} - \mu)^\intercal \Sigma^{-1}(\mathbf{w} - \mu)$$

$$\nabla \log p(\mathbf{w} \mid \mathbf{y}, X) = \frac{1}{\sigma^2}\phi_X(\mathbf{y} - \phi_X^\intercal \mathbf{w}) - \Sigma^{-1}(\mathbf{w} - \mu)$$

$$= (\sigma^{-2}\phi_X\phi_X^\intercal + \Sigma^{-1})\mathbf{w} - \sigma^{-2}\phi_X \mathbf{y} - \Sigma^{-1}\mu \overset{!}{=} 0$$

$$\mathbf{w}_* = (\sigma^{-2}\phi_X\phi_X^\intercal + \Sigma^{-1})^{-1}\left(\sigma^{-2}\phi_X \mathbf{y} + \Sigma^{-1}\mu\right)$$

$$\nabla\nabla^\intercal \log p(\mathbf{w} \mid \mathbf{y}, X) = \sigma^{-2}\phi_X\phi_X^\intercal + \Sigma^{-1}$$



$$
\begin{aligned}
p(\boldsymbol{w} \mid \boldsymbol{y}, X) &= \frac{1}{p(\boldsymbol{y})} \mathcal{N}(\boldsymbol{y} \mid \phi_X^{\intercal} \boldsymbol{w}, \sigma I) \mathcal{N}(\boldsymbol{w}; \mu, \Sigma) \\
&= \mathcal{N}(\boldsymbol{w}; w_* := \arg\min \mathcal{L}(\boldsymbol{y}, X), \nabla\nabla^{\intercal} \mathcal{L}(\boldsymbol{y}, X))
\end{aligned}
$$

- Gaussian linear regression is training a *single-layer* neural network with quadratic loss
- The minimum-loss weights of the neural network are the *posterior mean* (MAP estimate) of the Gaussian linear regression
- The curvature of the loss function is the *posterior covariance* of the Gaussian linear regression

We now have two options:

1. Add more features to the single layer
2. Add more layers / train the layers

Can we learn *functions* abstractly?

An observation from last lecture

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

We currently have

```python
@dataclasses.dataclass
class Gaussian:
    # Gaussian distribution with mean mu and covariance Sigma
    mu: jnp.ndarray  # shape (D,)
    Sigma: jnp.ndarray  # shape (D,D)
```

But we're trying to learn **functions** $y \approx f(x)$ from observations $(X, \boldsymbol{y}) \in (\mathbb{X} \times \mathbb{R})^n$

$$p(\boldsymbol{y} \mid f(X)) = \mathcal{N}(\boldsymbol{y}; f_X, \sigma^2 I) = \prod_{i=1}^{n} \mathcal{N}(y_i; f(x_i), \sigma^2)$$

So can we do something like this?

```python
@dataclasses.dataclass
class GaussianFunction:
    # mean function
    m: Callable[[jnp.ndarray], jnp.ndarray]  # maps |X^n -> |R^n
    # covariance function
    k: Callable[[jnp.ndarray, jnp.ndarray], jnp.ndarray]  # maps (|X x |X)^n -> |R^{n x n}
```

Note that “in practice”, functions are only defined through their values at finitely many concrete points

```
@dataclasses.dataclass
class GaussianProcess:
    # mean function
    m: Callable[[jnp.ndarray], jnp.ndarray]
    # covariance function
    k: Callable[[jnp.ndarray, jnp.ndarray], jnp.ndarray]

    def __call__(self, x):
        return Gaussian(mu=self.m(x), Sigma=self.k(x[:, None, :], x[None, :, :]))
```

What is **k**?

The Method of Least Squares

The Gaussian distribution is the unique choice yielding a mean that is the mean of measurements.

[image: C.A. Jensen, 1840]

J. Carl F. Gauss, 1777 Brunswick–1855 Göttingen

so wird allgemein sein müssen $\varphi'(M-p)+\varphi'(M'-p)+\varphi'(M''-p)+$ etc. $=0$, wenn für p der Werth $\frac{1}{\mu}(M+M'+M''+\text{etc.})$ substituirt wird, welches positive Ganzes nun auch durch μ ausdrückt sein mag. Setzt man daher voraus $M'=M''=$ etc. $=M-\mu N$, so wird allgemein, d. h. für jeden ganzen positiven Werth für μ, sein $\varphi'(\mu-1)N=(1-\mu)\varphi'(-N)$, woraus man leicht sieht, dass allgemein $\frac{\varphi'\Delta}{\Delta}$ eine constante Grösse sein müsse, welche ich mit k bezeichnen will. Hieraus wird $\log\varphi\Delta=\frac{1}{2}k\Delta\Delta+$ Const., oder wenn man die Basis der hyperbolischen Logarithmen mit e bezeichnet und die Constante $=\log \varkappa$ setzt,

$$\varphi\Delta=\varkappa e^{\frac{1}{2}k\Delta\Delta}.$$

Ferner sieht man leicht ein, dass k nothwendig negativ sein müsse, damit Ω in der That ein Grösstes werden könne, weshalb wir setzen $\frac{1}{2}k=-hh$; und da vermittelst des eleganten, zuerst von Laplace*) gefundenen Theorems das Integral $\int e^{-hh\Delta\Delta}d\Delta$, von $\Delta=-\infty$ bis zu $\Delta=+\infty$, wird $=\frac{\sqrt{\pi}}{h}$ (wobei π den halben Kreisumfang für den Radius $=1$ bezeichnet), so wird unsere Function werden:

$$\varphi\Delta=\frac{h}{\sqrt{\pi}}e^{-hh\Delta\Delta}.$$

178.

Die so eben ermittelte Function kann zwar nicht in aller Strenge die Wahrscheinlichkeiten der Fehler ausdrücken; denn da die möglichen Fehler
(213) stets in gewisse Grenzen eingezwängt sind, so müsste die Wahrscheinlichkeit grösserer Fehler immer $=0$ herauskommen, während unsere Formel stets einen begrenzten Werth darstellt. Dennoch aber ist dieser Mangel, an welchem jede analytische Function ihrer Natur nach laboriren muss, für jeden praktischen

*) In v. Zach „Monatliche Correspondenz" Band 21, S. 280 äussert Gauss: „Dass Euler schon das Theorem gefunden hat, woraus der schöne, von mir Laplace beigelegte Lehrsatz sehr leicht abgeleitet werden kann, fiel mir selbst schon früher ein, als aber die Stelle S. 212 schon abgedruckt war; ich wollte es aber nicht unter die Errata setzen, weil Laplace wenigstens das obige Theorem doch erst in der dort gebrauchten Form aufgestellt hat." *Anmerkung des Uebersetzers.*

Kernels

functions that build symmetric positive definite matrices

Definition (kernel)

$k : \mathbb{X} \times \mathbb{X} \rightarrow \mathbb{R}$ is a (Mercer / positive definite) **kernel** if, for any finite collection $X = [x_1, \dots, x_N]$, the matrix $k_{XX} \in \mathbb{R}^{N \times N}$ with $[k_{XX}]_{ij} = k(x_i, x_j)$ is **positive semidefinite**.

Definition (positive definite matrix)

A symmetric matrix $A \in \mathbb{R}^{N \times N}$ is called **positive (semi-) definite** if

$$v^\intercal A v \geq 0 \;\forall\, v \in \mathbb{R}^N.$$

Equivalently:

- All eigenvalues of the symmetric matrix A are non-negative
- A is a Gram matrix — the outer product of N vectors $[\phi_i]_{i=1,\dots,N}$

Gaussian processes

are **programs** that always produce valid Gaussians

Definition

Let $\mu : \mathbb{X} \to \mathbb{R}$ be any function, $k : \mathbb{X} \times \mathbb{X} \to \mathbb{R}$ be a Mercer kernel. A **Gaussian process** $p(f) = \mathcal{GP}(f; \mu, k)$ is a probability distribution over the function $f : \mathbb{X} \to \mathbb{R}$, such that every finite restriction to function values $f_X := [f_{x_1}, \dots, f_{x_N}]$ is a Gaussian distribution $p(f_X) = \mathcal{N}(f_X; \mu_X, k_{XX})$.

```python
@dataclasses.dataclass
class GaussianProcess:
    # mean function
    m: Callable[[jnp.ndarray], jnp.ndarray]
    # covariance function
    k: Callable[[jnp.ndarray, jnp.ndarray], jnp.ndarray]

    def __call__(self, x):
        return Gaussian(mu=self.m(x), Sigma=self.k(x[:, None, :], x[None, :, :]))
```

How do we build a kernel?

let's look at that algebra again

$$p(f(\bullet) \mid \mathbf{w}) = \mathcal{N}(f(\bullet); \phi(\bullet)^\intercal \mathbf{w}, \sigma I)$$

$$p(f(\bullet)) = \int p(f(\bullet) \mid \mathbf{w}) p(\mathbf{w}) \, \mathrm{d}\mathbf{w}$$

$$= \mathcal{N}(f(\bullet); \phi(\bullet)^\intercal \mu, \phi(\bullet)^\intercal \Sigma \phi(\circ) + \sigma I)$$

using the abstraction / encapsulation

$m(\bullet) := \phi(\bullet)^\intercal \mu$	$m : \mathbb{X} \rightharpoonup \mathbb{R}$	mean function
$k(\bullet, \circ) := \phi(\bullet)^\intercal \Sigma \phi(\circ)$	$k : \mathbb{X} \times \mathbb{X} \rightharpoonup \mathbb{R}$	covariance function, aka. **kernel**

What are we actually doing with those features?

let's look at that algebra again

$$
\begin{aligned}
p(f_\bullet \mid y, \phi_X) &= \mathcal{N}(f_\bullet; \phi_\bullet^\intercal \mu + \phi_\bullet^\intercal \Sigma \phi_X (\phi_X^\intercal \Sigma \phi_X + \sigma^2 I)^{-1} (y - \phi_X^\intercal \mu), \\
&\qquad \phi_\bullet^\intercal \Sigma \phi_\circ - \phi_\bullet^\intercal \Sigma \phi_X (\phi_X^\intercal \Sigma \phi_X + \sigma^2 I)^{-1} \phi_X^\intercal \Sigma \phi_\circ) \\
&= \mathcal{N}(f_\bullet; m_\bullet + k_{\bullet X} (k_{XX} + \sigma^2 I)^{-1} (y - m_X), \\
&\qquad k_{\bullet\circ} - k_{\bullet X} (k_{XX} + \sigma^2 I)^{-1} k_{X\circ})
\end{aligned}
$$

using the abstraction / encapsulation

$m_\bullet := \phi_\bullet^\intercal \mu$	$m : \mathbb{X} \rightharpoonup \mathbb{R}$	mean function
$k_{\bullet\circ} := \phi_\bullet^\intercal \Sigma \phi_\circ$	$k : \mathbb{X} \times \mathbb{X} \rightharpoonup \mathbb{R}$	covariance function, aka. **kernel**

Inner products are sums

an observation

- In feature-based (i.e. parametric) regression, we have

$$k_{\bullet\circ} := \phi_\bullet^\intercal \Sigma \phi_\circ = \sum_{i=1}^{F} \sum_{j=1}^{F} \phi(\bullet)_i \Sigma_{ij} \phi(\circ)_j$$

- For simplicity, let's fix $\Sigma = \frac{\sigma^2 (c_{\max} - c_{\min})}{F} I$, i.e. $\Sigma_{ij} = \frac{\sigma^2 (c_{\max} - c_{\min})}{F} \delta_{ij}$

thus: $$\phi(x_i)^\intercal \Sigma \phi(x_j) = \frac{\sigma^2 (c_{\max} - c_{\min})}{F} \sum_{i=1}^{F} \sum_{j=1}^{F} \phi(x_i)^2 \delta_{ij} \phi(x_j)^2 = \frac{\sigma^2 (c_{\max} - c_{\min})}{F} \sum_{\ell=1}^{F} \phi_\ell(x_i) \phi_\ell(x_j)$$

- Sometimes sums can be finite even if they contain infinitely many terms…

$$\sum_{i=0}^{\infty} x^i = \frac{1}{1-x} \qquad \text{for} \quad |x| < 1$$

Sometimes, **more** features make things **cheaper**

an example

[DJC MacKay, 1998]

- For simplicity, let's fix $\Sigma = \frac{\sigma^2(c_{\max}-c_{\min})}{F} I$, i.e. $\Sigma_{ij} = \frac{\sigma^2(c_{\max}-c_{\min})}{F}\delta_{ij}$

  thus: $$\phi(x_i)^\intercal \Sigma \phi(x_j) = \frac{\sigma^2(c_{\max}-c_{\min})}{F} \sum_{i=1}^{F}\sum_{j=1}^{F} \phi(x_i)^2 \delta_{ij} \phi(x_j)^2 = \frac{\sigma^2(c_{\max}-c_{\min})}{F} \sum_{\ell=1}^{F} \phi_\ell(x_i)\phi_\ell(x_j)$$

- especially, for $\phi_\ell(x) = \exp\left(-\frac{(x-c_\ell)^2}{2\lambda^2}\right)$

$$\begin{aligned}
\phi(x_i)^\intercal \Sigma \phi(x_j) &= \frac{\sigma^2(c_{\max}-c_{\min})}{F} \sum_{\ell=1}^{F} \exp\left(-\frac{(x_i-c_\ell)^2}{2\lambda^2}\right) \exp\left(-\frac{(x_j-c_\ell)^2}{2\lambda^2}\right) \\
&= \frac{\sigma^2(c_{\max}-c_{\min})}{F} \exp\left(-\frac{(x_i-x_j)^2}{4\lambda^2}\right) \sum_{\ell}^{F} \exp\left(-\frac{(c_\ell-\frac{1}{2}(x_i+x_j))^2}{\lambda^2}\right)
\end{aligned}$$

Sometimes, **more** features make things **cheaper**

an example

[DJC MacKay, 1998]

$$\phi(x_i)^\intercal \Sigma \phi(x_j) = \frac{\sigma^2 (c_{\max} - c_{\min})}{F} \exp\left(-\frac{(x_i - x_j)^2}{4\lambda^2}\right) \sum_{\ell}^{F} \exp\left(-\frac{(c_\ell - \frac{1}{2}(x_i + x_j))^2}{\lambda^2}\right)$$

now increase F so # of features in δc approaches $\frac{F \cdot \delta c}{(c_{\max} - c_{\min})}$

$$\phi(x_i)^\intercal \Sigma \phi(x_j) \to \sigma^2 \exp\left(-\frac{(x_i - x_j)^2}{4\lambda^2}\right) \int_{c_{\min}}^{c_{\max}} \exp\left(-\frac{(c - \frac{1}{2}(x_i + x_j))^2}{\lambda^2}\right) dc$$

let $c_{\min} \to -\infty$, $c_{\max} \to \infty$

$$k(x_i, x_j) := \phi(x_i)^\intercal \Sigma \phi(x_j) \to \sqrt{\pi} \lambda \sigma^2 \exp\left(-\frac{(x_i - x_j)^2}{4\lambda^2}\right)$$

$$k_{x_i x_j} = \sqrt{\pi} \lambda \sigma^2 \exp\left(-\frac{(x_i - x_j)^2}{4\lambda^2}\right)$$

Demo

Our construction is enough

It is fine to think of them as inner products

Lemma: k is a Mercer kernel if it can be written as (assuming $\mathcal{L}$ is positive set for ν)

$$k(x,x') = \sum_{\ell \in \mathcal{L}} \phi_\ell(x)\phi_\ell(x') \quad \text{or} \quad = \int_{\mathcal{L}} \phi_\ell(x)\phi_\ell(x')\, d\nu(\ell)$$

Proof: $\forall X \in \mathbb{X}^N, v \in \mathbb{R}^N : v^\intercal k_{XX} v = \int \sum_i^N v_i \phi_\ell(x_i) \sum_j^N v_j \phi_\ell(x_j)\, d\nu(\ell) = \int \left[\sum_i v_i \phi_\ell(x_i)\right]^2 d\nu(\ell) \geq 0$ $\square$

Theorem (Mercer, 1909): (precise form in Lecture 11). *If* k is a Mercer kernel, *then* it can be written as above (with *countably* many terms).

All kernels admit a (potentially infinite) feature expansion

Summary: Gaussian Processes

- Sometimes it is possible to consider **infinitely many** features at once, by extending from a sum to an integral. This requires some regularity assumption about the features' locations, shape, etc.
- The resulting **nonparametric model** is known as a **Gaussian process**
- Inference in GPs is tractable (though at polynomial cost $\mathcal{O}(N^3)$ in the number N of datapoints)

Please cite this course, as

```
@techreport{Tuebingen_ProbML23,
    title =
    {Probabilistic Machine Learning},
    author = {Hennig, Philipp},
    series = {Lecture Notes
        in Machine Learning},
    year = {2023},
    institution = {Tübingen AI Center}}
```