Gaussian Processes (Contd)

Piyush Rai

Topics in Probabilistic Modeling and Inference (CS698X)

Jan 30, 2019
Announcement

- Quiz 1 tomorrow - Jan 31, 7pm-8pm
- Y14, Y15, Y18: RM-101
- Y16, Y17: KD-101
- Bring a pencil and eraser (answers to be written on the question paper itself)
- Do not bring anything else
Recap: Bayesian Modeling of Nonlinear Functions

- Goal: Learn a nonlinear function $f$ for discriminative models of the form $p(y|x)$, e.g.,
  
  $$
  p(y|f, x) = \mathcal{N}(y|f(x), \beta^{-1})
  $$

  $$
  p(y|f, x) = [\sigma(f(x))]^y[1 - \sigma(f(x))]^{1-y}
  $$

  $$
  p(y|f, x) = \text{ExpFam}(f(x))
  $$

- Not just interested in a point estimate but the full posterior over $f$

- Usually done in one of the following ways
  - Ad-hoc: Define nonlinear features $\phi(x) +$ train Bayesian linear model ($f(x) = w^T \phi(x)$))
  - Ad-hoc: Train a neural net to extract features $\phi(x) +$ train Bayesian linear model
  - Bayesian Neural Networks (infer posterior over NN weights; compute posterior predictive)
  - Gaussian Processes (Bayesian modeling + kernels)
Recap: Gaussian Process

- A Gaussian Process is a distribution over functions
- Denoted as $\mathcal{GP}(\mu, \kappa)$; parametrized by a mean function $\mu$ and covariance/kernel function $\kappa$

![Graph showing Gaussian Process](image)

- Mean function $\mu$ models the “average” function $f$ from $\mathcal{GP}(\mu, \kappa)$: $\mu(x) = \mathbb{E}_{f \sim \mathcal{GP}(\mu, \kappa)}[f(x)]$
- Cov. function $\kappa$ models “shape/smoothness” of functions from this GP
  - $\kappa(.,.)$ is a function that computes similarity between two inputs
Recap: Gaussian Process

- For $f \sim \mathcal{GP} (\mu, \kappa)$, $f$'s values at any finite set of input $x_1, \ldots, x_N$ are jointly Gaussian

$$
\begin{bmatrix}
  f(x_1) \\
  f(x_2) \\
    \vdots \\
  f(x_N)
\end{bmatrix}
\sim \mathcal{N}
\begin{bmatrix}
  \mu(x_1) \\
  \mu(x_2) \\
    \vdots \\
  \mu(x_N)
\end{bmatrix}
, 
\begin{bmatrix}
  \kappa(x_1, x_1) & \cdots & \kappa(x_1, x_N) \\
  \kappa(x_2, x_1) & \cdots & \kappa(x_2, x_N) \\
    \vdots & \ddots & \vdots \\
  \kappa(x_N, x_1) & \cdots & \kappa(x_N, x_N)
\end{bmatrix}
$$

- In a more compact notation, $p(f) = \mathcal{N}(\mu, K)$, where $f$ and $\mu$ are $N \times 1$ and $K$ is $N \times N$
- Can use it to easily compute $f_\ast = f(x_\ast)$ for a new input $x_\ast$. To see this, note that for $\mu = 0$

$$
p\left( \begin{bmatrix} f \\ f_\ast \end{bmatrix} \right) = \mathcal{N}\left( \begin{bmatrix} f \\ f_\ast \end{bmatrix} \middle| \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} K & k_\ast \\ k_\ast^\top & \kappa(x_\ast, x_\ast) \end{bmatrix} \right)
$$

where $k_\ast = [\kappa(x_\ast, x_1), \ldots, \kappa(x_\ast, x_N)]^\top$

- Can now apply the Gaussian conditioning to get $p(f_\ast | f) = \mathcal{N}(\mu_\ast, \sigma_\ast^2)$ where

$$
\begin{align*}
\mu_\ast &= k_\ast^\top K^{-1} f = \sum_{n=1}^{N} w_n f_n = \sum_{n=1}^{N} \alpha_n k(x_n, x_\ast) \\
\sigma_\ast^2 &= \kappa(x_\ast, x_\ast) - k_\ast^\top K^{-1} k_\ast
\end{align*}
$$

Prob. Mod. & Inference - CS698X (Piyush Rai, IITK)  Gaussian Processes (Contd)
Some functions drawn from a **GP prior**
(note: Blue dots are values of a randomly drawn function at a small number of inputs; the solid curves are generated by evaluating the functions at a large # of inputs)

Some functions drawn from the **GP posterior**
after observing 5 \((x,f(x))\) pairs
GP: Noiseless to Noisy Setting

- In many cases, we are modeling outputs $y_n$ that are “noisy” versions of $f_n = f(x_n)$, e.g.,

$$p(y_n | f_n) = \mathcal{N}(y_n | f_n, \beta^{-1})$$
$$p(y_n | f_n) = [\sigma(f_n)]^{y_n} [1 - \sigma(f_n)]^{1-y_n}$$
$$p(y_n | f_n) = \text{ExpFam}(f_n)$$

- Here making predictions for a new input $x_*$ requires not $p(f_* | f)$ but $p(y_* | y)$

$$p(y_* | y) = \int p(y_* | f_*) p(f_* | y) df_* = \int p(y_* | f_*) p(f_*) p(f | y) df df_*$$

- For the above, $p(y_* | f_*)$ and $p(f | y) \propto p(f) p(y | f)$ will depend on likelihood model $p(y_n | f_n)$

- However $p(f_* | f)$ will be the same as in the noiseless setting (i.e., a Gaussian as we saw) :-)

- Note: For GP Regression (with Gaussian noise), $p(y_* | y)$ is very easily computable!
The likelihood model: \( p(y|f) = \mathcal{N}(y|f, \sigma^2 I_N) \). The prior distribution: \( p(f) = \mathcal{N}(f|0, K) \)

The posterior predictive \( p(y_*|x_*, y, X) \) or \( p(y_*|y) \) (skipping \( X, x_* \) from the notation) will be

\[
p(y_*|y) = \int p(y_*|f_*) p(f_*|y) df_* = \int p(y_*|f_*) p(f_*|f) p(f|y) df df_*
\]

where all the 3 distributions in the integrand are Gaussians in case of GP regression!

Therefore it is an easy to compute integral!

However, we can compute \( p(y_*|y) \) even without using the above method

Reason: The *marginal distribution* of the training data responses \( y \)

\[
p(y) = \int p(y|f)p(f)df = \mathcal{N}(y|0, K + \sigma^2 I_N) = \mathcal{N}(y|0, C_N)
\]

Using the same result, the marginal distribution \( p(y_*) = \mathcal{N}(y_*|0, \kappa(x_*, x_*) + \sigma^2) \)
GP Regression: Making Predictions

- Let's consider the joint distr. of $N$ training responses $y$ and test response $y^*$

$$p\begin{pmatrix} y \\ y^* \end{pmatrix} = \mathcal{N}\begin{pmatrix} y \\ y^* \end{pmatrix} \bigg| \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \mathbf{c}_N & \mathbf{k}_*^\top \\ \mathbf{k}_* & c \end{pmatrix}$$

where $\mathbf{k}_* = [\kappa(x^*, x_1), \ldots, \kappa(x^*, x_N)]^\top$, $c = \kappa(x^*, x^*) + \sigma^2$

- The desired predictive posterior will be (using conditional from joint property of Gaussian)

$$p(y^* | y) = \mathcal{N}(y^* | \mu^*, \sigma^2_{*})$$

$$\mu^* = \mathbf{k}_*^\top \mathbf{C}_N^{-1} y$$

$$\sigma^2_{*} = \kappa(x^*, x^*) + \sigma^2 - \mathbf{k}_*^\top \mathbf{C}_N^{-1} \mathbf{k}_*$$

- Note that this is almost identical to the noiseless case (with $\sigma^2$ added to the predictive variance)

- Can interpret predictive mean $\mu^*$ as kernelized SVM or nearest neighbor based prediction
GP Regression: An Illustration

Red curve: True function
Red points: Noisy training examples
Black curve: Predictive mean
Shaded part: Predictive variance
GP Regression: Learning Hyperparameters

- There are two hyperparameters in the GP regression model
  - Variance of the Gaussian noise $\sigma^2$
  - Assuming $\mu = 0$, the hyperparameters $\theta$ of the covariance/kernel function $\kappa$, e.g.,
    \[
    \kappa(x_n, x_m) = \exp\left(-\frac{|x_n - x_m|^2}{\gamma}\right) \quad \text{(RBF kernel)}
    \]
    \[
    \kappa(x_n, x_m) = \exp\left(-\sum_{d=1}^{D} \frac{(x_{nd} - x_{md})^2}{\gamma_d}\right) \quad \text{(ARD kernel)}
    \]
    \[
    \kappa(x_n, x_m) = \kappa_{\theta_1}(x_n, x_m) + \kappa_{\theta_2}(x_n, x_m) + \ldots + \kappa_{\theta_M}(x_n, x_m) \quad \text{(flexible composition of multiple kernels)}
    \]

- Type-II MLE is a popular choice for learning these hyperparams, by maximizing marginal likelihood
  \[
p(y|\sigma^2, \theta) = \mathcal{N}(y|0, \sigma^2 I_N + K_\theta)
  \]

- MLE-II for GP regression maximizes the log marginal likelihood w.r.t. the hyperparameters
  \[
  \log p(y|\sigma^2, \theta) = -\frac{1}{2} \log |\sigma^2 I_N + K_\theta| - \frac{1}{2} y^\top (\sigma^2 I_N + K_\theta)^{-1} y + \text{const}
  \]
Binary classification: Now the likelihood $p(y|f)$ will be Bernoulli: $p(y_n|f_n) = \text{Bernoulli}(\sigma(f_n))$

For multiclass GP ($K > 2$ classes), $p(y_n|f_n)$ will be multinoulli (note: $f_n$ will be a $K \times 1$ vector)

For GP GLM, $p(y_n|f_n)$ will be some exp-family distribution

The prior is still GP, therefore $p(f) = \mathcal{N}(0, K)$

The posterior predictive $p(y_*|y)$ can again be written as

$$p(y_*|y) = \int p(y_*|f_*)p(f_*|y)df_* = \int p(y_*|f_*)p(f_*|f)p(f|y)df df_*$$

This in general is not as easy to compute as in case of GP regression

- $p(f_*|f)$ is still not a problem (will be Gaussian)
- $p(f|y) \propto p(f)p(y|f)$ will require approximation (e.g., Laplace, MCMC, variational, etc.)
- The overall integral will require approximation as well
Scalability Aspects of GP

- Computational costs in some steps of GP based models scale in the size of training data
  - E.g., test time prediction in GP regression takes $O(N)$ time

$$p(y_*|y) = \mathcal{N}(y_*|\mu_*, \sigma_*^2)$$

$$\mu_* = k_*^\top C_N^{-1} y$$  \hspace{1cm} (O(N) cost assuming $C_N^{-1}$ is pre-computed)

$$\sigma_*^2 = k(x_*, x_*) + \sigma^2 - k_*^\top C_N^{-1} k_*$$

- GP models often require matrix inversions - takes $O(N^3)$ time. Storage also requires $O(N^2)$ space

- A lot of work on speeding up GPs\(^1\). Some approaches for speeding up GPs
  - Inducing Point Methods (condition the predictions only on a small set of “learnable” points)
  - Divide-and-Conquer methods (learn GP on small subsets of data and aggregate predictions)
  - Kernel approximations

- Note that nearest neighbor methods and kernel methods also face similar issues w.r.t. scalability
  - Many tricks to speed up kernel methods can be used for speeding up GPs too

---

\(^1\) When Gaussian Process Meets Big Data: A Review of Scalable GPs - Liu et al, 2018
GP: A few comments

- GP is a nonparametric model. Why called “nonparametric”?  
  Complexity (representation size) of the function \( f \) grows in the size of training data  
  To see this, note the form of the GP predictions, e.g., predictive mean in GP regression

\[
\mu_* = f(x_*) = k_*^\top C_N^{-1} y = k_*^\top \alpha = \sum_{n=1}^{N} \alpha_n k(x_*, x_n)
\]

- It implies that \( f(\cdot) = \sum_{n=1}^{N} \alpha_n k(\cdot, x_n) \), which means \( f \) is written in terms of all training examples  
- Thus the representation size of \( f \) depends on the number of training examples

- In contrast, a parametric model has a size that doesn’t grow with training data  
  E.g., a linear model learns a fixed-sized weight vector \( w \in \mathbb{R}^D \) (\( D \) parameters, size independent of \( N \))

- Nonparametric models therefore are more flexible since their complexity is not limited beforehand  
  Note: Methods such as nearest neighbors and kernel SVMs are also nonparametric (but not Bayesian)

- GPs equivalent to infinitely-wide single hidden-layer neural net (under some technical conditions)
Neural Networks and Gaussian Processes

- An infinitely-wide single hidden layer NN with i.i.d. priors on weights = a Gaussian Process
- Shown formally by (Radford Neal, 1994)$^2$. Based on a simple application of central limit theorem

A useful result for several reasons

- Can use a GP instead of an infinitely wide Bayesian NN (which is impractical anyway)
- With GPs, inference is easy (at least for regression and with known hyperparams)
- A proof that GPs can also learn any function (just like infinitely wide neural nets - Hornik’s theorem)

Connection recently generalized to infinitely wide multiple hidden layer NN (Lee et al, 2018)$^3$

---

$^2$Priors for infinite networks, Tech Report, 1994
$^3$Deep Neural Networks as Gaussian Processes (ICLR 2018)
GP: A few other comments

- Can be thought of as Bayesian analogues of kernel methods
  - Can get estimate in the uncertainty in the function and its predictions

- Can learn the kernel (by learning the hyperparameters of the kernels)

- Not limited to supervised learning problems
  - The function $f$ could even be a mapping of an unknown quantity to an observed quantity

\[ x_n = f(z_n) + \text{“noise”} \]

where $z_n$ is a latent representation of $x_n$ ("GP latent variable models" for nonlin. dim. red.)

- Many mature implementations of GP exist. You may check out
  - GPML (MATLAB), GPsuff (MATLAB/Octave), GPy (Python), GPyTorch (PyTorch)
Other Recent Advances on Gaussian Processes

- Deep Gaussian Processes (DGP)
  - Akin to a deep neural network where each hidden node is modeled by a GP

- A nice alternative to linear transform + nonlinearity based neural nets, e.g., $h = \tanh(Wx)$

- GPs with deep kernels defined by neural nets

- Neural Processes and Conditional Neural Processes (GP + neural nets): Most recent development
GPs are very versatile!

- GPs enable us to learn nonlinear functions while also capturing the uncertainty.

- Uncertainty can tell us where to acquire more training data to improve the function’s estimate.
  - Especially useful if we can’t get too many training examples (e.g., expensive inputs and/or labels).

- This is very useful in a wide range of applications involving sequential decision-making.
  - **Active Learning**: Learning a function by gathering the most informative training examples.
  - **Bayesian Optimization**: Optimizing an expensive to evaluate functions (and maybe we don’t even know it form) – boils down to simultaneous function learning and optimization.
Bayesian Optimization: The Basic Formulation

- Consider finding the optima $x_*$ (say minima) of a function $f(x)$

- Caveat: We don’t know the form of the function; can’t get its gradient, Hessian, etc
- Suppose we can only query the function’s values at certain points (i.e., only black-box access)
- Thus we have to learn the function as well as find its optima
- Can learn the function using GP and use the uncertainty to decide which $f(x)$ value to query next
- Will look at it in more detail later this semester