Gaussian Process Model for Time Series Prediction

Sergey V.

2024

1 Introduction

Gaussian Process Regression (GPR) is a non-parametric Bayesian approach to regression. It provides both a predicted mean function and an uncertainty measure, making it particularly useful in time series forecasting, for applications where uncertainty is crucial. For example, financial time series data, such as FOREX markets or in scientific applications.

2 Background: Gaussian Processes

A Gaussian Process (GP) is a generalization of the multivariate Gaussian distribution to infinite-dimensional spaces. A GP defines a distribution over functions, fully specified by its mean function $m(x)$ and covariance (kernel) function $k(x, x')$. Formally, a GP can be written as:

$$f(x) \sim \mathcal{GP}(m(x), k(x, x')).$$

2.1 Mean and Covariance Functions

The mean function $m(x)$ represents the expected value of the function at any point x:

$$m(x) = \mathbb{E}[f(x)].$$

In practice, the mean function is often assumed to be zero for simplicity:

$$m(x) = 0.$$

The covariance function $k(x, x')$ encodes the relationship between pairs of input points x and x'. It defines the shape and smoothness of the functions drawn from the GP:

$$k(x, x') = \mathrm{Cov}(f(x), f(x')).$$

2.2 Constructing the GP Prior

The GP prior construction is the first step in defining a Gaussian Process model. It lays down the fundamental assumptions about how the data points are correlated with one another. This prior does not depend on the observations (y_train); instead, it defines the model's assumptions purely based on the inputs (x_train). A Gaussian Process is a collection of random variables, any finite number of which have a joint Gaussian distribution. To construct a GP prior, we need two components: Mean Function, $\mu(x)$ and Covariance Function (Kernel), $k(x_i, x_j)$. This mean function defines the expected output at each input point before seeing any data. It's often assumed to be zero ($\mu(x) = 0$) for simplicity and because the GP can model the data's underlying pattern regardless of the initial mean. The kernel function captures the relationships or correlations between different input points. It encodes assumptions about the data's smoothness, periodicity, and other structural properties. Common kernels include the Gaussian, Matern, Periodic, and linear kernels. Given a set of training inputs $X = \{x_1, \ldots, x_n\}$ and corresponding outputs $y = \{y_1, \ldots, y_n\}$, we assume:

$$f(X) \sim \mathcal{N}(0, K(X, X)),$$

where $K(X, X)$ is the covariance matrix computed using the kernel function, with entries $K_{ij} = k(x_i, x_j)$.

3 Prediction with Gaussian Processes

Given the training data (X, y) and a set of test inputs $X_* = \{x_1^*, \ldots, x_m^*\}$, the goal is to predict the function values $f(X_*)$ at the test inputs. We assume a joint Gaussian distribution over the observed and predicted data:

$$\begin{bmatrix} f(X) \\ f(X_*) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} K(X, X) & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix} \right).$$

Here: - $K(X, X)$ is the covariance matrix of the training data. - $K(X, X_*)$ is the covariance matrix between training and test points. - $K(X_*, X) = K(X, X_*)^\top$ is the transpose of the covariance between test and training points. - $K(X_*, X_*)$ is the covariance matrix of the test points.

3.1 Conditional Distribution for Predictions

The predictive distribution for the test points $f(X_*)$ conditioned on the observed data (X, y) is a Gaussian:

$$f(X_*)|X, y, X_* \sim \mathcal{N}(\mu_*, \Sigma_*),$$

where:

$$\mu_* = K(X_*, X)K(X, X)^{-1}y,$$
$$\Sigma_* = K(X_*, X_*) - K(X_*, X)K(X, X)^{-1}K(X, X_*).$$

Step-by-step Explanation:

- **Step 1:** Compute the covariance matrices using the kernel function:
  - $K(X, X)$ for training data.
  - $K(X_*, X)$ between training and test points.
  - $K(X_*, X_*)$ for the test points.
- **Step 2:** Invert the training covariance matrix $K(X, X)$ to incorporate the information provided by the observed data.
- **Step 3:** Compute the predictive mean μ_* by weighting the observations y using the covariance between the training and test points.
- **Step 4:** Compute the predictive variance Σ_* to quantify the uncertainty of the predictions.

Additional details: Given a set of training data (X, y) and a set of test inputs X_*, the goal of Gaussian Process regression is to predict the function values at X_*. This involves computing both the predictive mean and variance, which quantify the model's predictions and uncertainty.

1. Constructing the Covariance Matrices
- Training Covariance Matrix: $K(X, X)$ is an $n \times n$ matrix that represents the covariance between all pairs of training inputs:

$$K(X, X)_{ij} = k(x_i, x_j),$$

where n is the number of training points, and $k(\cdot, \cdot)$ is the chosen kernel function.
- Cross-Covariance Matrix: $K(X_*, X)$ is an $m \times n$ matrix that represents the covariance between each test input X_* and each training input X:

$$K(X_*, X)_{ij} = k(x_{*,i}, x_j).$$

- Test Covariance Matrix: $K(X_*, X_*)$ is an $m \times m$ matrix representing the covariance between the test inputs:

$$K(X_*, X_*)_{ij} = k(x_{*,i}, x_{*,j}),$$

where m is the number of test points.

2. Predictive Mean Formula
The predictive mean for the test inputs is given by:

$$\mu_* = K(X_*, X)K(X, X)^{-1}y.$$

Step-by-Step Breakdown
1. Compute Cross-Covariance: $K(X_*, X)$ quantifies the relationship between each test point and the training points. 2. Compute Covariance Inverse: $K(X, X)^{-1}$ is the inverse of the training covariance matrix. Since this matrix is often large and potentially ill-conditioned, it is more efficiently computed using Cholesky decomposition. 3. Linear Combination: The term $K(X_*, X)K(X, X)^{-1}$ forms a linear combination of the observed data y. This step adjusts the test

points based on the correlation (as expressed in the kernel) with the training data. 4. Predictive Mean: The resulting vector μ_* provides the GP's best estimate of the function values at the test inputs X_*.

3. Predictive Variance Formula
The predictive variance is given by:

$$\Sigma_* = K(X_*, X_*) - K(X_*, X)K(X, X)^{-1}K(X, X_*).$$

Step-by-Step Breakdown
1. Initial Variance: $K(X_*, X_*)$ represents the variance of the test inputs based solely on the kernel function. This is the prior uncertainty before considering the training data. 2. Covariance Reduction: The term $K(X_*, X)K(X, X)^{-1}K(X, X_*)$ represents the part of the variance that can be "explained" by the training data. This reduction accounts for the knowledge gained from observing the training points. 3. Final Variance: Subtracting this reduction from $K(X_*, X_*)$ gives the posterior variance Σ_*. This variance quantifies the uncertainty in the predictions after observing the training data.

4 Cholesky Decomposition and Efficient Computation

Direct inversion of the covariance matrix $K(X, X)$ can be computationally expensive and numerically unstable. To address this, Cholesky decomposition is used:

$$K(X, X) = LL^\top,$$

where L is a lower triangular matrix. The inverse can then be computed as:

$$K(X, X)^{-1} = (L^\top)^{-1}L^{-1}.$$

5 Kernel Functions

The choice of kernel function is crucial in GPR as it defines the shape and smoothness of the functions that the GP can represent. In this work, we use a combination of several kernel functions:

5.1 Matern Kernel

$$k_{\text{Matern}}(x, x') = a\left(1 + \frac{\sqrt{3}|x - x'|}{b}\right)\exp\left(-\frac{\sqrt{3}|x - x'|}{b}\right).$$

5.2 Periodic Kernel

$$k_{\text{Periodic}}(x, x') = a\exp\left(-\frac{2\sin^2\left(\frac{\pi|x - x'|}{c}\right)}{b^2}\right).$$

5.3 Quadratic Kernel

$$k_{\text{Quadratic}}(x, x') = a\left(1 + \frac{|x - x'|^2}{2cb^2}\right)^{-c}.$$

5.4 Linear Kernel (Trend Correction)

$$k_{\text{Linear}}(x, x') = \lambda(x \cdot x').$$

5.5 Combined Kernel

$$k_{\text{Combined}}(x, x') = k_{\text{Matern}}(x, x') + k_{\text{Periodic}}(x, x') + k_{\text{Quadratic}}(x, x') + k_{\text{Linear}}(x, x').$$

6 Gaussian Process Prediction Calculations

Given training data (X, y) and test inputs X_*, the predictive distribution of the function values at X_* is:

$$f(X_*)|X, y, X_* \sim \mathcal{N}(\mu_*, \Sigma_*),$$

where:

$$\mu_* = K(X_*, X)K(X, X)^{-1}y,$$
$$\Sigma_* = K(X_*, X_*) - K(X_*, X)K(X, X)^{-1}K(X, X_*).$$

6.1 Explanation of Terms

- $K(X, X)$: The covariance matrix of the training data. - $K(X_*, X)$: The covariance matrix between the test points and training data. - $K(X_*, X_*)$: The covariance matrix of the test points. - $K(X, X)^{-1}$: The inverse of the training covariance matrix, which is computed efficiently using Cholesky factorization. - μ_*: The predictive mean, which represents the best estimate of the function's value at the test points. - Σ_*: The predictive variance, which quantifies the uncertainty in the predictions.

The function `gp_solve` implements the above Gaussian Process Regression.

```
gp_solve <- function(x_train, y_train, x_pred, kernel, sigma2e, a, b, c) {
  # Compute the covariance matrix for the training data
  K_train_train <- kernel(x_train, x_train, a, b, c)

  # Add observation noise for numerical stability
  K_train_train <- K_train_train + sigma2e * diag(nrow(K_train_train))

  # Compute covariance between training and prediction points
  K_train_pred <- kernel(x_train, x_pred, a, b, c)
  K_pred_pred <- kernel(x_pred, x_pred, a, b, c)
  # Add jitter for numerical stability to K_pred_pred
  K_pred_pred <- K_pred_pred + sigma2e * diag(nrow(K_pred_pred))

  # Cholesky decomposition for numerical stability
  L <- chol(K_train_train)
  L_inv <- solve(L)

  # Compute the inverse of K_train_train
  K_train_inv <- t(L_inv) %*% L_inv

  # Compute the predictive mean
  alpha <- K_train_inv %*% y_train
  mu_pred <- t(K_train_pred) %*% alpha

  # Compute the predictive variance
  cov_pred <- K_pred_pred - t(K_train_pred) %*% K_train_inv %*% K_train_pred

  # Return the mean and variance
  solution <- list(mu = mu_pred, var = cov_pred)
  return(solution)
}
```

Explanation of `gp_solve`:
1. Covariance Matrix Calculation**: The covariance matrices $K_{\text{train,train}}$, $K_{\text{train,pred}}$, and $K_{\text{pred,pred}}$ are computed using the specified kernel function. 2. Add Observation Noise: Noise term σ_e^2 is added to the diagonal of the training covariance matrix to ensure numerical stability. 3. Cholesky Decomposition: The matrix $K_{\text{train,train}}$ is decomposed using Cholesky factorization into a lower triangular matrix L, which is used to efficiently compute its inverse. 4. Predictive Mean: Calculated as $\mu_* = K_{\text{train,pred}}^\top K_{\text{train,train}}^{-1} y$. 5. Predictive Variance: Calculated as $\Sigma_* = K_{\text{pred,pred}} - K_{\text{train,pred}}^\top K_{\text{train,train}}^{-1} K_{\text{train,pred}}$. 6. Return Solution: The function returns the predicted mean and variance.

7 Log-Likelihood for Parameter Optimization: `get_log_likelihood`

The log-likelihood function measures how well the GP model fits the data given the kernel parameters:

$$\log p(\mathbf{y}_{\text{train}}|\mathbf{X}_{\text{train}}, \theta) = -\frac{1}{2}\mathbf{y}_{\text{train}}^T \mathbf{K}^{-1} \mathbf{y}_{\text{train}} - \frac{1}{2}\log|\mathbf{K}| - \frac{n}{2}\log 2\pi,$$

where $\mathbf{K} = k(\mathbf{X}_{\text{train}}, \mathbf{X}_{\text{train}}) + \sigma_n^2 \mathbf{I}$. The goal is to maximize this log-likelihood with respect to the kernel parameters $\theta = (a, b, c)$. The function `get_log_likelihood` computes the log-likelihood of the data given the GP model.

```
get_log_likelihood <- function(x_train, y_train, kernel, sigma2e, a, b, c) {
  # Compute the covariance matrix
  k.xx <- kernel(x_train, x_train, a, b, c)

  # Add noise for numerical stability
  V <- k.xx + sigma2e * diag(nrow(k.xx))

  # Cholesky decomposition
  chol_V <- tryCatch(chol(V), error = function(e) NULL)
  if (is.null(chol_V)) return(Inf)

  # Invert the covariance matrix using Cholesky
  Vinv <- solve(t(chol_V)) %*% solve(chol_V)

  # Log-likelihood calculation
  log_likelihood <- -0.5 * t(y_train) %*% Vinv %*% y_train
  log_likelihood <- log_likelihood - sum(log(diag(chol_V)))
  log_likelihood <- log_likelihood - (length(y_train) / 2) * log(2 * pi)

  return(as.numeric(log_likelihood))
}
```

Explanation of `get_log_likelihood`:
1. Covariance Matrix: Computes the covariance matrix $K_{\text{train,train}}$ using the specified kernel. 2. Add Noise: Adds a noise term σ_e^2 to ensure numerical stability. 3. Cholesky Decomposition: Uses Cholesky decomposition to factorize $K_{\text{train,train}}$ into a lower triangular matrix, which is used for efficient inversion. 4. Log-Likelihood Calculation: - The first term, $-\frac{1}{2}y^\top K_{\text{train,train}}^{-1}y$, measures how well the model fits the data. - The second term, $-\frac{1}{2}\log\det(K_{\text{train,train}})$, penalizes model complexity. - The third term, $-\frac{n}{2}\log(2\pi)$, is a normalization constant. 5. Return Log-Likelihood: Returns the computed log-likelihood, which is used in hyperparameter optimization.

8 Data Scaling, Trend Correction, and Prediction

8.1 Data Scaling

Before applying the GP, the data is centered and scaled to ensure numerical stability and better model performance:

$$y_{\text{scaled}} = \frac{y - \text{mean}(y)}{\text{std}(y)}.$$

After prediction, the results are unscaled using the stored mean and standard deviation:

$$y_{\text{unscaled}} = y_{\text{predicted}} \times \text{std}(y) + \text{mean}(y).$$

To predict future prices, the input space x_{test} is extended beyond the observed data range. This extension allows the GP to make predictions based on the prior data's learned patterns.

8.2 Trend Correction

The combined kernel includes a linear kernel that captures the overall trend of the data. This term is critical in financial time series where prices have an underlying drift.

After computing the GP predictions, the linear trend can be added back:

$$y_{\text{final}} = y_{\text{GP}} + \lambda(x_{\text{test}} \times \text{slope}) + \text{intercept}.$$

9 Conclusion

This document outlines the construction and use of Gaussian Process Regression for univariate time series. The approach utilizes a combined kernel to capture both oscillatory patterns and underlying trends. The log-likelihood is used to optimize kernel hyperparameters, ensuring the best fit for the data. The implementations of `gp_solve` and `get_log_likelihood` demonstrates the mechanics of making predictions, estimating model uncertainty, and evaluating kernel parameter choices.