Gaussian Process Homework and Solutions

Final Report – Undergraduate Group 3

Mario Gutierrez, Tyler Chan, Eric Zhong

CSCI 4962 - Machine Learning and Optimization, Fall 2023

Objective of this assignment The objective of this assignment is to strengthen your proficiency in Gaussian processes, particularly in the context of regression tasks. You will delve into mathematical derivations related to multivariate normal distributions, covariance matrices, and posterior distributions. Additionally, you will apply this knowledge to implement a Gaussian Process Regression model using the Radial Basis Function (RBF) kernel. Practical aspects of the assignment include importing and preprocessing a dataset, implementing key functions for Gaussian Process Regression, and tuning hyperparameters through grid search. The assignment culminates in the application of the developed model to real-world data, with a focus on evaluating performance metrics and visualizing predictions. This comprehensive exploration aims to deepen your understanding of Gaussian processes and their practical application in regression tasks.

Instructions Create a Jupyter notebook for this assignment, and use Python 3. Write documented, readable and clear code (e.g. use reasonable variable names). Your code should run on the TA's computer, but do not expect them to run it to produce the output; instead, submit it with the outputs shown. Submit this notebook interspersing any textual answers in markdown cells (using LaTeX), clearly labeled, along with your code.

1. **Problem 1**: Deriving Multinormal Distribution (25pts)

In this exercise, we will validate a fundamental concept discussed in our pre-recorded lecture. We have two datasets: one with n training examples, denoted as X and y , along with another dataset consisting of m test examples, labeled as X^* and y^* Let $0_n$and 0_mrepresent zero vectors with lengths corresponding to n and m, respectively. Additionally, let k denote a specific kernel function.

If the datasets have the following normal distribution:

$$\begin{bmatrix} y \\ y^* \end{bmatrix} = \mathcal{N}_{\begin{bmatrix} y \\ y^* \end{bmatrix}}\left(\begin{bmatrix} 0_n \\ 0_m \end{bmatrix}, \begin{bmatrix} k(X,X) & k(X,X^*) \\ k(X^*,X) & k(X^*,X^*) \end{bmatrix}\right)$$

Like in the lecture, you will assume that the dataset has multivariate normal distribution.

a. Derive and show that the mean of the datasets can be calculated as:
$\mu = k(X^*,X)k(X^*,X)^{-1}y$

Hint: The Conditional Distribution of X_1 given known values of X_2 is a multivariate norm with:

$$\mu = \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(X_2 - \mu_2)$$

b. Derive and show that the variance of the dataset can be calculated as:
$\Sigma = k(X^*,X) - k(X^*,X)k(X^*,X)^{-1}k(X,X^*)$.

Hint: Start by finding the Covariance between the following linear transformation and y.

$$z = y^* - k(X^*,X)k(X,X)^{-1}y$$

c. Derive and show that the posterior distribution can be calculated as:
$\mathrm{P}(y^*|X^*,\ X,y) = \mathcal{N}_{y^*}(\mu,\Sigma)$

Problem 1 Solution:

a. From Lecture 3 in class, we learned that the mean of a random variable X for a normal distribution is:

$$\mu = \mathbb{E}[X]$$

Given that: $\mathbb{E}[y^*] = 0$ and $\mathbb{E}[y] = 0$ from the problem statement.
The covariance $\Sigma_{y,y} = k(X,X)$ is given by the covariance matrix between the training set y.
The covariance $\Sigma_{y^*,y} = k(X^*,X)$ is given by the covariance matrix between the training set y.

From the hint based on Gaussian conditional distribution we can compute the mean μ as:

$$\mu = \mathbb{E}[y^*] + \Sigma_{y,y}\Sigma_{y^*,y}^{-1}(y - \mathbb{E}[y^*]) = 0 + \Sigma_{12}\Sigma_{22}^{-1}(y-0) = \Sigma_{12}\Sigma_{22}^{-1}(X_2) = k(X^*,X)k(X,X)^{-1}y$$

$$\boxed{\mu = k(X^*,X)k(X,X)^{-1}y}$$

b. To prove this, three small properties must be proved and built upon:

For simplicity, let $\boldsymbol{B} = k(X^*,X)k(X,X)^{-1}$ in the given linear transformation:
$$z = y^* - k(X^*,X)k(X,X)^{-1}y$$

i. The covariance between z and y is given by:

$$Cov(z,y) = Cov(y^* - \boldsymbol{B}y, y) = Cov(y^*, y) - Cov(\boldsymbol{B}y, y) = Cov(y^*, y) - \boldsymbol{B}Cov(y,y)$$

Because z and y are linearly independent, we get:

$0 = Cov(y^*, y) - \boldsymbol{B}Var(y,y)$, Thus: $\underline{Cov(y^*, y) = \boldsymbol{B}Var(y,y) = \boldsymbol{B}k(X,X)}$

ii. When trying to solve $\Sigma = \mathrm{Var}(y^*|y)$ we can replace y^* for z to get $\Sigma = Var(z - \boldsymbol{B}y|y)$

From variance basic properties, we get:

$$\Sigma = Var(z|y) + Var(\boldsymbol{B}y|y) - \boldsymbol{B}\ Cov(z, -y) - Cov(z, -y)\boldsymbol{B}^T$$

But the conditional variance between y and y is zero. Also, the $Cov(z, -y)\boldsymbol{B}^T$ from what we learned in **b.i**.

Thus: $\underline{\Sigma = Var(z|y) = Var(z)}$ since the values of y are known and z and y are independent.

iii. $\Sigma = Var(z|y) = Var(z) = Var(y^* - \boldsymbol{B}y)$

$\Sigma = Var(y^*) - \boldsymbol{B}Var(y)\boldsymbol{B^T} + \boldsymbol{B}Cov(y^*, y) - Cov(y^*, y)\boldsymbol{B}^T$

But: $Var(y^*) = k(X^*, X^*)$ and $Var(y) = k(X,X)$

And: $Cov(y^*, y) = \boldsymbol{B}Var(y,y) = \boldsymbol{B}k(X,X)$ from **b.i**

Thus: $\Sigma = k(X^*, X^*) - \boldsymbol{B}k(X,X)\boldsymbol{B^T} + \boldsymbol{BB}k(X,X) - \boldsymbol{B}k(X,X)\ \boldsymbol{B}^T$

$\Sigma = k(X^*, X^*) - 2\boldsymbol{B}k(X,X)\boldsymbol{B^T} + \boldsymbol{BB}k(X,X)$

Replacing $\boldsymbol{B} = k(X^*, X)k(X,X)^{-1}$ we get:

$$\Sigma = k(X^*, X^*) - 2k(X^*, X)\cancel{k(X,X)}^{-1}\cancel{k(X,X)}k(X,X^*)k(X,X)^{-1} + k(X^*, X)k(X,X)^{-1}k(X^*, X)k\cancel{(X,X)}^{-1}\cancel{k(X,X)}$$

$$\Sigma = k(X^*, X^*) - 2k(X^*, X)k(X,X^*)k(X,X)^{-1} + k(X^*, X)k(X,X)^{-1}k(X^*, X)$$

Thus: $\boxed{\Sigma = k(X^*, X^*) - k(X^*, X)k(X,X^*)k(X,X)^{-1}}$

c. The posterior distribution $P(y^*|X^*, X, y)$ is obtained by conditioning the joint $P(y^*|y)$ on the observed data (X, y). Using the properties of multivariate normal distributions, we can express the posterior distribution as:

$$P(y^*|X^*, X, y) = \mathcal{N}_{y^*}(\mu, \Sigma)$$

From Bayes Theorem: $P(y^*|X^*, X, y) = \frac{P(X*,X,y|y^*)*P(y^*|y)}{P(X*,X,y)}$

This is because, the likelihood function $P(X*, X, y|y^*)$ is a constant with respect to y^*. Similarly, $P(X*, X, y)$ does not need to be explicitly calculated as because it involves integrating over all possible values of y and y^* , and it serves as a normalizing constant in Bayes' Theorem.

Thus:

$$P(y^*|X^*, X, y) = \mathcal{N}_{y^*}(\mu, \Sigma)$$

Where μ and Σ have already been derived from problem sections **1.a** and **1.b**.

2. **Problem 2**: Gaussian Process Regression (50pts)

Although Gaussian Process Regression models have been typically used for time-series data, they may also be employed for other more traditional applications. Here, you will implement a basic Gaussian Process Regression model using the Radial Basis Function (RBF) kernel:

$$k(x_i, x_j) = \sigma e^{\left(-\frac{||x_i - x_j||_2^2}{2\ell^2}\right)}$$

Where σ is the weight and ℓ is the length scale hyperparameters.

We will test your implementation on a dataset found in the Compressive Failure of Concrete dataset found in the UCI repository. This dataset has eight (8) features and one label, 'Concrete Compressive Strength (MPa)'. The dataset has been provided as 'Concrete_Data.xls'.

a. Import the dataset, split it into a 70% training set and a 30% test set. Label these as X_train, y_train, X_test, and y_test. Use the sklearn.preprocessing function StandardScaler to fit_transform the X_train and transform the X_test datasets. Be sure not to scale the labels.

b. Write the Radial Basis Function (RBF) kernel as function K_1 = RBFKernel(X1, X2, sigma, Ls). This function should accept two matrices of examples, along with hyperparameters sigma and h, and generates the kernel matrix K, where each element $k(x_i, x_j)$ is computed using the RBF function.
Where sigma is the weight and Ls is the length scale hyperparameters. Set sigma and Ls equal to 1 for now.
Compute: K = rbf_kernel(X_train, X_train, sigma, Ls); print(K.shape)

c. Write the Gaussian Process Regression function [GP_Mean, GP_Var] = GP_Reg_Pred(X_train,y_train,X_test,gamma,sigma,Ls), which performs the regression task and returns the mean and variances (predictions) for the random variable X_test. Use the algorithm found in Rasmus and Williams Chapter 2 (page 19). Implement the Cholesky factorization using scipy.linalg.cholesky function.
Where gamma is the noise parameter for the Gaussian Process specifically. Set gamma=1e-5 for now.
Compute: mean_test_posterior, var_test_posterior = gp_Reg_Pred(X_train, y_train, X_test, gamma, sigma, Ls);
print(np.mean(mean)); print(np.var(mean))

d. To fit the hyperparameters, we must first compute our objective function (log marginal likelihood). Write a function [LMHood] = LogMarLhood (X_train, y_train, gamma, sigma,Ls) that does this. Use the algorithm found in Rasmus and Williams Chapter 2 (page 19).
Compute: LMHood = LogMarLhood(X_train, y_train, gamma, sigma, Ls);
print(LMHood);

e. Write a function [best_Ls, best_sigma, best_gamma] = Grid_Search (X_train, y_train, Lss, sigmas, gammas), which does a grid search across the gammas, hs, sigmas. These input arguments represent the combination of parameters that minimizes the log marginal likelihood. Additionally, set gamma to be $gammas * \sigma_y$ where σ_y is the standard deviation of the training set. The function should iterate through all the possible combination of hyperparameters and provide the combination of hyperparameters that minimizes the log marginal likelihood.

f. Execute your Gaussian process regression algorithm on the X_train, y_train, X_test, y_test dataset. Obtain your hyperparameters by employing your implemented Grid_Search function and exploring the parameter space using:
Lss = np.logspace(-1, 1, 10)*np.linalg.norm(np.std(X_train))
gammas = np.logspace(-1, 1, 10)
sigmas = np.logspace(-1, 1, 10)*np.std(y_train)
Compute: print(best_Ls, best_sigma, best_gamma).

g. Calculate the Mean Absolute error and R^2 between the train and test set. Similarly, plot the train and test set as a scatterplot

Disclosure: The minimization of the negative log-marginal-likelihood is not typically done using Grid Search, this was just done in this example to show how useful Gaussian Processes can be despite being a non-parametric machine learning method. Typically, the log-marginal-likelihood is typically minimized using quasi-Newtonian methods. For further information on quasi-Newtonian methods, see Lecture 30 by Dr. Steven Johnson.

Problem 2 Solution: See Jupyter Notebook with solution.

3. **Problem 3**: Sampling from the GP prior and posterior

In this exercise, we'll be writing the necessary code to illustrate and plot samples of f from a Gaussian process prior using a squared exponential (or, equivalently, radial basis function) kernel.

$$f \sim GP(m, \kappa) \text{ Where: } m(x) = 0 \; and \; \mathrm{k}(\mathrm{x_i}, \mathrm{x_j}) = \sigma \mathrm{e}^{\left(-\frac{||\mathrm{x_i}-\mathrm{x_j}||_2^2}{2\ell^2}\right)}$$

To execute this process, we select a vector of m test input points, denoted as x∗. The choice of x∗ is made to include an ample number of points, ensuring that it forms a visually continuous line on the screen. Subsequently, we assess the m × m covariance matrix $k(x*, x*)$ and proceed to generate samples from the multivariate normal distribution.

$$f(x*) \sim \mathcal{N}(m(x*), k(x*, x*))$$

Generating Priors

a. Employ numpy's linspace function to construct a vector x* with m=100 elements that are evenly distributed throughout -4 and 4.
b. Create a mean vector, denoted as m(x∗), consisting of 101 elements, all set to zero. Moreover, generate a 101 × 101 covariance matrix, κ(x∗, x∗), using the provided expression for $k(x_i, x_j)$. Set the hyperparameters as ℓ = 2 and σ = 1.
c. Use scipy's multivariate_normal function from the stats module to generate 25 new samples denoted as $f^{(1)}(x*), \ldots, f^{(25)}(x*)$, from the multivariate normal distribution represented by $f(x*) \sim \mathcal{N}(m(x*), k(x*, x*))$.
d. Plot the 25 samples versus the input vector x*.
e. Experiment with a different value of ℓ and iterate through steps (b) to (d). Compare the resulting plots and identify the distinctions between the two.

Generating Posterior Predictions

In these subtask, you will implement Gaussian Process Regression (GPR) using scikit-learn and evaluate the model's performance.

f. Generate synthetic data for the target label using the mean of the 25 samples generated in subtask c and the noise term found below:
```
noise = np.random.normal(loc=0, scale=noise_level, size=len(x*))
```
Split the label and feature vectors into a training and test set. Test set should be the beginning 50% of the total dataset.
g. Use scikit-learn's GaussianProcessRegressor with a Radial Basis Function (RBF) kernel and a White Kernel for noise. Train the model using the training data. Functions needed:
sklearn.gaussian_process.GaussianProcessRegressor
sklearn.gaussian_process.kernels.RBF
sklearn.gaussian_process.kernels.WhiteKernel
Plot the predicted mean along with the 95% confidence interval (mean ± 1.96 * std) for the test set. Ensure that the plot includes the actual test data points.
h. Bonus: Split the dataset such that the training set covers the first 30% and last 30% of the data. Repeat section g. What do you see?

For this problem feel free to experiment with different values for the noise level (noise_level) and observe how it affects the model predictions and confidence intervals. Similarly, experiment with different ratios of training and test set sizes!

Problem 3 Solution: See Jupyter Notebook with solution.