ML HW5-1 Report

309552007 袁鈺勛

A. Code

a. Mode

There are two modes in the program. Mode 0 is gaussian process without optimization. And mode 1 is gaussian process with optimization. It can be controlled by –m parameter in command.

```
# Start gaussian process
if not mode:
    info_log('=== Without optimization ===')
    gaussian_process(x, y, n)
else:
    info_log('=== With optimization ===')

    # Get optimized alpha and length scale
    info_log('Get optimized parameters')
    guess = np.array([1.0, 1.0])
    res = minimize(marginal_log_likelihood, guess)

    opt_alpha, opt_length_scale = res.x
    gaussian_process(x, y, n, opt_alpha, opt_length_scale)
```

b. Kernel

It is a rational quadratic kernel with variance 1. I use cdist to calculate the distance between each pair.

```
# variance * (1 + d(xi, xj)^2 / 2al^2)^(-a)
return 1.0 * np.power(1 + cdist(x_i, x_j, 'sqeuclidean') / (2 * alpha * length_scale * length_scale), -alpha)
```

c. Gaussian process

The implementation of gaussian process is based on the solution in the following image.

$$\mu(\mathbf{x}^*) = k(\mathbf{x}, \mathbf{x}^*)^\top \mathbf{C}^{-1}\mathbf{y}$$

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

$$k^* = k(\mathbf{x}^*, \mathbf{x}^*) + \beta^{-1}$$

Firstly, it calculates the covariance matrix (or called kernel) from training data. Then it computes the kernel between training data and testing data and the kernel between testing data itself. Based on the results, it can calculate the mean and variance of the conditional distribution.

```
# Get testing data
num_of_points = 1000
x_test = np.linspace(-60, 60, num_of_points).reshape(-1, 1)

# Get covariance matrix
info_log('Get covariance matrix')
covariance = rational_quadratic_kernel(x_coord, x_coord, alpha, length_scale)

# Get kernel of testing data to testing data
info_log('Get kernel of testing data to testing data')
k_test = np.add(rational_quadratic_kernel(x_test, x_test, alpha, length_scale), np.eye(len(x_test)) / noise)

# Get kernel of training data to testing data
info_log('Get kernel of training data to testing data')
k_train_test = rational_quadratic_kernel(x_coord, x_test, alpha, length_scale)

# Get mean and variance
info_log('Get mean and variance')
mean = k_train_test.T.dot(np.linalg.inv(covariance)).dot(y_coord).ravel()
variance = k_test - k_train_test.T.dot(np.linalg.inv(covariance)).dot(k_train_test)
```

d. Marginal

According to the log-likelihood in the following image, it can find the optimal hyperparameters from negative marginal log-likelihood.

$$p(\mathbf{y}|\theta) = \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{C}_\theta)$$

$$\ln p(\mathbf{y}|\theta) = -\frac{1}{2}\ln |\mathbf{C}_\theta| - \frac{1}{2}\mathbf{y}^\top \mathbf{C}_\theta^{-1}\mathbf{y} - \frac{N}{2}\ln(2\pi) \;\text{☞}\; \frac{\partial \ln p(\mathbf{y}|\theta)}{\partial \theta}$$

```
# Get covariance matrix
covariance = rational_quadratic_kernel(x, x, alpha=theta[0], length_scale=theta[1])

# - ln p(y|θ) = 0.5*ln|C| + 0.5*y^T*C^(-1)*y + N/2*ln(2π)
return 0.5 * np.log(np.linalg.det(covariance)) + 0.5 * y.ravel().T.dot(np.linalg.inv(covariance)).dot(
    y.ravel()) + points / 2.0 * np.log(2.0 * np.pi)
```

e. Optimization

The optimized hyperparameters are found by minimizing marginal log-likelihood. And I use an initial guess of two 1.0.

```
# Get optimized alpha and length scale
info_log('Get optimized parameters')
guess = np.array([1.0, 1.0])
res = minimize(marginal_log_likelihood, guess)

opt_alpha, opt_length_scale = res.x
gaussian_process(x, y, n, opt_alpha, opt_length_scale)
```

f. Graph

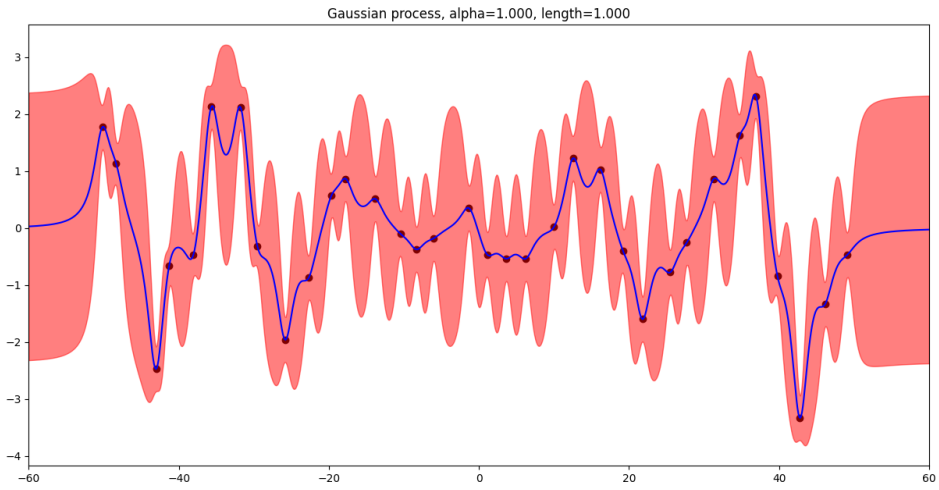
Because of 95% confidence, the interval is 1.96 times variance. Then I can scatter all the training points and draw the mean with blue line and interval with red region.

```
# Get 95% confidence upper and lower bound
info_log('Get confidence interval')
upper_bound = mean + 1.96 * variance.diagonal()
lower_bound = mean - 1.96 * variance.diagonal()

# Draw the graph
info_log('Draw the graph')
plt.xlim(-60, 60)
plt.title(f'Gaussian process, alpha={alpha:.3f}, length={length_scale:.3f}')
plt.scatter(x_coord, y_coord, c='k')
plt.plot(x_test.ravel(), mean, 'b')
plt.fill_between(x_test.ravel(), upper_bound, lower_bound, color='r', alpha=0.5)
plt.tight_layout()
plt.show()
```

B. Results

a. Randomly picked hyperparameters



b. Optimized hyperparameters

C. Observations

a. Optimization version is better than random version.

b. In the interval that there are training data, gaussian process can make prediction with higher confidence. However, it can hardly predict the outcome in the interval that there isn't any training point.

c. Variance in the kernel function isn't important. It doesn't severely affect the prediction. It only represents how sparse the function is.