ML Homework 05-1: Gaussian Process

- Student ID: 309553002
- Name: 林育愷

ML Homework 05-1: Gaussian Process
- Code with Detailed Explanations
  - Prerequisites
  - Kernel Function
  - Prediction
  - Minimization
  - Visualization
- Experiments Settings and Results
- Observations and Discussion
- References

Code with Detailed Explanations

Prerequisites

I use Python 3.6 for this implementation, with the following packages

- NumPy,
- SciPy, and
- Matplotlib.

Kernel Function

I refer to the following equation

$$k(x_a, x_b) = \sigma^2 \left(1 + \frac{\left\|x_a - x_b\right\|^2}{2\alpha\ell^2}\right)^{-\alpha}$$

to construct a rational quadratic kernel function[1], where σ means the overall variance, l means the length scale, and α means the scale-mixture (with $\alpha > 0$). In this term parameters of the Gaussian process would be $\theta = (\sigma, \alpha, l)$.

```
class GaussianProcess(object):
    @staticmethod
    def kernel_function(xa, xb, sigma, alpha, l):
        val = 1 + (xa - xb)**2 / (2 * alpha * l**2)
        val = sigma**2 * np.power(val, -alpha)
        return val
```

Prediction

Recall that Gaussian process is a data-driven method, so from here we assume that there is a set of training data

$$\{(x_i, y_i)\}_{i=1}^{N} = (X, Y)$$

where the model is written as $y_i \sim \mathcal{N}(f_\theta(x_i), \beta^{-1})$. In this homework is stored as `input.data` so in the beginning we load them as two arrays.

```python
def load_data(filename):
    data = []
    with open(filename, 'r') as f:
        for line in f:
            data.append(list(map(float, line.split())))
    data = np.array(data, dtype=float)
    return data[:, 0], data[:, 1]
```

Now for each x_0 with given θ, the predicted mean μ and variance σ^2 of $y_0 \sim \mathcal{N}(\mu, \sigma^2)$ is said to be

$$\mu(x_0) = k(X, x_0)^T C^{-1} Y$$
$$\sigma^2(x_0) = k(X, X) + \beta^{-1} - k(X, x_0)^T C^{-1} k(X, x_0)$$

where $C_\theta = k(X, X) + \beta^{-1} I \in \mathbb{R}^{N \times N}$ [2].

> For vectorized calculation we will put all x's together as $X^\star$, and only get the diagonal part of the covariance matrix.

```python
import numpy as np

class GaussianProcess(object):
    def __init__(self, x, y, beta, sigma, alpha, l):
        self.x = x
        self.y = y
        self.beta = beta
        self.beta_inv = 1 / beta
        self.sigma = sigma
        self.alpha = alpha
        self.l = l

    @property
    def kernel_matrix(self):
        x = self.x
        xb = np.tile(x, x.shape).reshape(x.shape + (-1, ))
        xa = xb.T
        return self.kernel_function(xa, xb, self.sigma, self.alpha, self.l)

    @property
    def covariance_matrix(self):
        cov = self.kernel_matrix + self.beta_inv * np.eye(self.x.shape[0])
        return cov

    @property
    def inv_covariance_matrix(self):
        return np.linalg.inv(self.covariance_matrix)

    @property
    def negative_log_likelihood(self):
        x, y = self.x, self.y
        cov = self.covariance_matrix
        val = 0.5 * np.log(np.linalg.det(cov))
        val += 0.5 * y @ np.linalg.inv(cov) @ y
        val += x.shape[0] / 2 * np.log(2 * np.pi)
```

```
            return val

    def predict(self, xs):
        sigma = self.sigma
        alpha = self.alpha
        l = self.l
        beta_inv = self.beta_inv

        xb = np.tile(xs, self.x.shape).reshape(self.x.shape + (-1, ))
        xa = np.tile(self.x, xs.shape).reshape(xs.shape + (-1, )).T
        kern = self.kernel_function(xa, xb, sigma, alpha, l)
        mean = kern.T @ self.inv_covariance_matrix @ self.y

        xb_star = np.tile(xs, xs.shape).reshape(xs.shape + (-1, ))
        xa_star = xb_star.T
        kern_star = self.kernel_function(xa_star, xb_star, sigma, alpha, l)
        var = kern_star + beta_inv - kern.T @ self.inv_covariance_matrix @ kern
        std = np.sqrt(np.diag(var))

        return mean, std
```

Minimization

For optimization of the Gaussian process, I directly use `scipy.optimize.minimize` function to solve the following program with respect to negative log likelihood function [2]

$$\arg\min_{\theta=(\sigma,\alpha,l)} \frac{1}{2}\log\det(C_\theta) + \frac{1}{2}Y^T C_\theta^{-1} + \frac{N}{2}\log(2\pi).$$

Refer to the source code of SciPy, the algorithm I used is the BFGS algorithm (Broyden–Fletcher–Goldfarb–Shanno algorithm).

```
import numpy as np
import scipy.optimize as opt

class GaussianProcess(object):
    def minimize(self):
        def energy_function(x):
            self.sigma = x[0]
            self.alpha = x[1]
            self.l = x[2]
            return self.negative_log_likelihood

        x = np.array([self.sigma, self.alpha, self.l])
        res = opt.minimize(energy_function, x)
        # trigger the function again to surely store the minimizer
        energy_function(res.x)
```

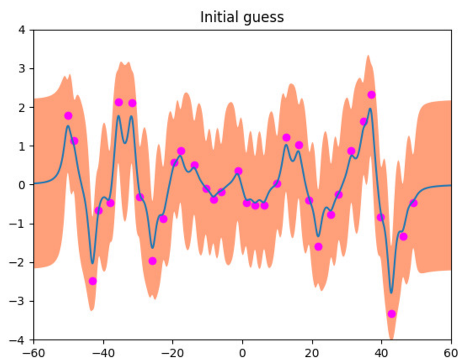
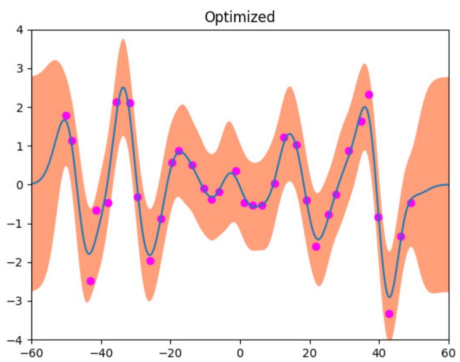
Visualization

95% confidence interval for Gaussian distribution means the 2-sigma bound. So for given $X^\star = \{-60, \ldots, 60\}$ and the corresponding predicted result $\mu(Y^\star)$ and $\sigma(Y^\star)$ with respect to θ, we visualized the result with $(X^\star, \mu(Y^\star))$, $(X^\star, \mu(Y^\star) + 2\sigma(Y^\star))$, and $(X^\star, \mu(Y^\star) - 2\sigma(Y^\star))$.

```python
import matplotlib.pyplot as plt

def visualize(x, y, xs, ymean, ystd, title=None):
    plt.figure(title)
    plt.title(title)
    plt.plot(xs, ymean)
    plt.fill_between(xs,
                     ymean + 2 * ystd,
                     ymean - 2 * ystd,
                     facecolor='lightsalmon')
    plt.ylim(-4, 4)
    plt.xlim(-60, 60)
    plt.scatter(x, y, color='magenta')
```

Experiments Settings and Results

With my initial guess of θ being $(1, 1, 1)$, and $\beta = 5$, follow the above approach the experiment results is shown below.

Initial guess $\theta = (1, 1, 1)$	**Optimized result** $\theta^{\star} = (1.311, 495.880, 3.316)$
	

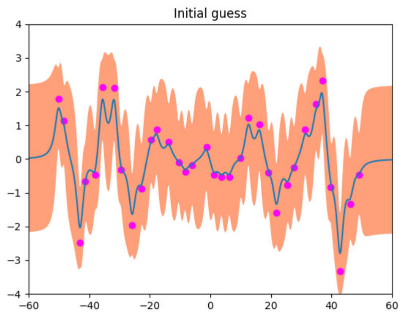
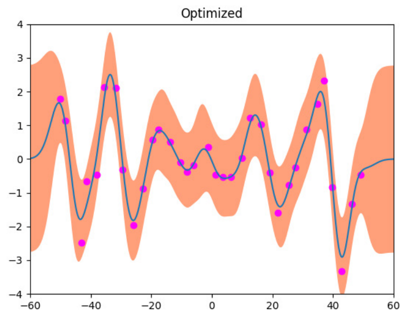
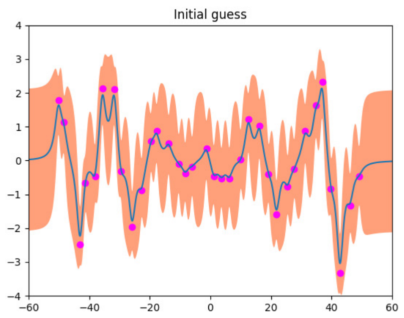
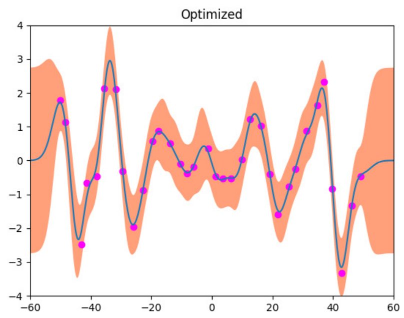
We can observe that after optimization the sigma bound is tighter than initial guess in general.

```python
INPUT_FILENAME = './input.data'
BETA = 5
SIGMA_INIT = 1
ALPHA_INIT = 1
L_INIT = 1

if __name__ == '__main__':
    x, y = load_data(INPUT_FILENAME)

    gp = GaussianProcess(x, y, BETA, SIGMA_INIT, ALPHA_INIT, L_INIT)

    xs = np.linspace(-60, 60, 1000)
    ymean, ystd = gp.predict(xs)
    visualize(x, y, xs, ymean, ystd, title='Initial guess')

    # minimize parameter sigma, alpha, and l
    gp.minimize()

    ymean, ystd = gp.predict(xs)
```

```
    visualize(x, y, xs, ymean, ystd, title='Optimized')
    plt.show()
```

Observations and Discussion

With higher β, we are able to get tighter result than previous section (theoretically it means that the input data is more reliable).

	Initial guess	Optimized result
$\beta = 5$		
$\beta = 10$		

References

1. https://peterroelants.github.io/posts/gaussian-process-kernels/#Rational-quadratic-kernel. ↩

2. Slides of this course. ↩ ↩