309551101 郭育麟

ML_HW05

1 . Gaussian Process

1-1:

Kernel function : Rational quadratic kernel

$$(1+\frac{(x_i-x_j)^2}{2*alpha*Length\_scale^2})^{-alpha}$$

Alpha : 1 , Length_scale = 1

In this homework , first using cal_kernel() to calculate data points' similarity and return a kernel matrix . Then use predict() function to evaluate the new points (-60,60) each mean and variance . Finally use plot_gaussian() function to draw the result .

Data points (raw data) are sample from

$$y_n = f(x_n) + \epsilon_n$$

$$where\ \varepsilon_n \sim N(0, B^{-1})$$

$$\mathrm{B} = 5$$

Cal_kernel():

```
def cal_kernel(x1,x2,alpha,length_scale):
    '''
    using rational quadratic kernel function: k(x_i, x_j) = (1 + (x_i-x_j)^2 / (2*alpha * length_scale^2))^-alpha
    :param X1: (n) ndarray
    :param X2: (m) ndarray
    return: (n,m)  ndarray
    '''
    x1 = x1.reshape(-1,1)
    x2 = x2.reshape(1,-1)
    temp = np.power(x1-x2,2)
    kernel = np.power(1+temp/(2*alpha-length_scale**2),-alpha)
    return kernel
```

$$(1+\frac{(x_i-x_j)^2}{2*alpha*Length\_scale^2})^{-alpha}$$

Predict():

```
def predict(x_line , X, y ,K ,beta, alpha , length_scale,cal_kernel):
    '''
    vectorize calculate k_x_xstar !!
    :param x_line: sampling in linspace(-60,60)
    :param X:  (n) ndarray
    :param y: (n) ndarray
    :param K: (n,n) ndarray
    :param beta:
    :return: (len(x_line),1) ndarray, (len(x_line),len(x_line)) ndarray
    '''
    k_x_xstar = cal_kernel(X,x_line,alpha,length_scale)
    k_xstar_xstar = cal_kernel(x_line,x_line,alpha,length_scale)
    means = np.dot(np.dot(k_x_xstar.T,np.linalg.inv(K)),y.reshape(-1,1))
    k_star =  k_xstar_xstar+ (1/beta)
    variances = k_star - np.dot(np.dot(k_x_xstar.T,np.linalg.inv(K)),k_x_xstar)
    return means,variances
```

$$\mathrm{C} = \mathrm{k} + \mathrm{B}^{-1} * I$$

$$k^* = k(x^*, x^*) + B^{-1}$$

$$\text{means}(x^*) = k(x,x^*)^T * C^{-1} * y$$
$$\text{variance}^2 = k^* - k(x,x^*)^T * C^{-1} * k(x,x^*)$$

plot_gaussian():

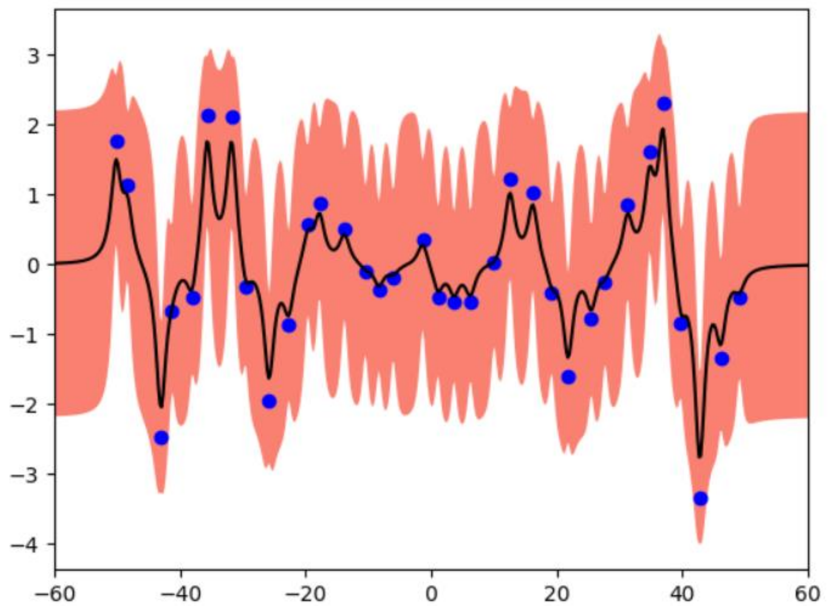
```
def plot_gaussian(x,y,x_line,mean_predict,variance_predict,name):
    plt.plot(x,y,'bo')
    plt.plot(x_line,mean_predict,'k-')
    plt.fill_between(x_line,mean_predict+2*variance_predict,mean_predict-2*variance_predict,facecolor='salmon')
    plt.xlim(-60,60)
    plt.savefig(name+".png")
```

Given raw data ,mean and variance can plot the graph , 95% confidence interval means that $\text{mean} \pm 2 * \text{standard deviation}$.

Main():

```
###5.1
path = './input.data'
beta = 5
alpha = 1
length_scale = 1
name = "hw5"
x , y = dataloader(path)
kernel = cal_kernel(x,x,alpha,length_scale)+1/beta*np.identity(len(x))
x_line=np.linspace(-60,60,num=500)
mean_predict,variance_predict = predict(x_line , x, y ,kernel ,beta, alpha , length_scale,cal_kernel)
mean_predict=mean_predict.reshape(-1)
variance_predict = np.sqrt(np.diag(variance_predict))
plot_gaussian(x,y,x_line,mean_predict,variance_predict,name)
```

Result:



In the blue dots (raw data) interval , the variance would smaller than new points (-60~60) . Raw data has more specific range than new points .

1-2:

In this exercise , we need to optimize our parameter(alpha and lengh_scale) first . I use scipy.optimize minimize the negative likelihood function . After minimize I can get the better parameters so it can predict result same as 1-1 .

Initially function:

$$\frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} e^{\frac{-1}{2}(x-\mu)^T\Sigma(x-\mu)}$$

After differential could get objective function .

Objective function :

$$\ln P(y|\theta) = \frac{-1}{2}\ln|C_\theta| - \frac{-1}{2}y^T C_\theta^{-1} y - \frac{N}{2}\ln(2\pi)$$

Objective_function():

```
def objective_function(X,y,beta):
    '''
    :param X:  (n) ndarray
    :param y:  (n) ndarray
    :param beta:
    :return:
    '''
    def objective(theta):
        K = cal_kernel(X,X,alpha=theta[0],length_scale=theta[1])+(1/beta)*np.identity(len(X))

        target1 = np.dot(np.dot(0.5*y.reshape(1,-1),np.linalg.inv(K)),y.reshape(-1,1))

        target2 = 0.5*np.log(np.linalg.det(K))

        target3 = 0.5*len(X)*np.log(2*np.pi)

        target = target1 + target2+target3
        return target

    return objective
```

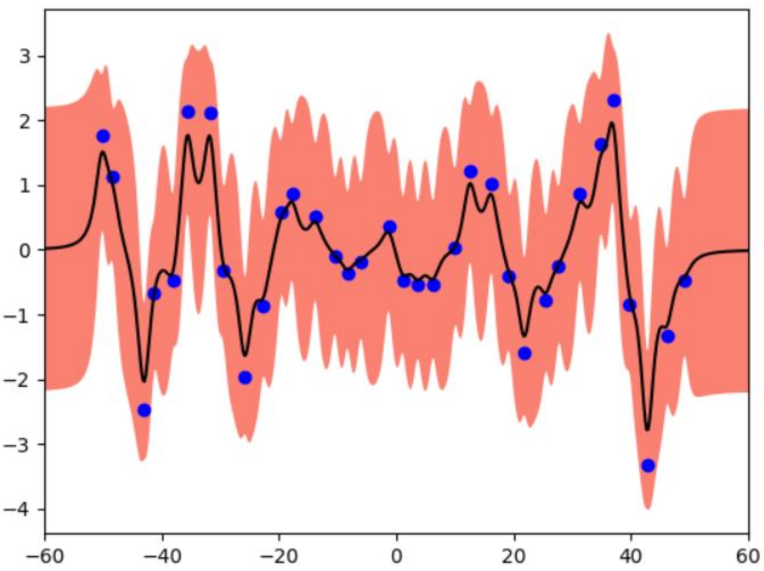
In this function , calculate the objective function value .

Minimize_negative_marginal():

```
def minimize_negative_margianl(X,y,beta):
    objective_value = 1e9
    inits= [1e-2,1e-1,0,1e1,1e2]
    for init_alpha in inits:
        for init_length_scale in inits:
            res=minimize(objective_function(X,y,beta),x0=[init_alpha,init_length_scale],bounds=((1e-5,1e5),(1e-5,1e5)))
            if res.fun<objective_value:
                objective_balue = res.fun
                alpha_optimize,length_scale_optimize=res.x
    print("Optimize alpha is %.3f Optimize length_scale is %.3f"%(alpha_optimize,length_scale_optimize))
    return alpha_optimize,length_scale_optimize
```

In this function , initial sets parameters from [1e-2,1e-1,1e1,1e2] and want to minimize the objective function . The reason why choose [1e-2 , 1e-1, 1e1 , 1e2] is avoid the minimize value is local minimum not global minimum . Finally , return alpha and length_scale .

Result:



Compare:

In result 1-2 , the variance in blue dot (raw data) becomes smaller . The parameters which has been optimized would let the likelihood be maximum . So it can has better performance than 1-1