I. Gaussian Process

a. code with detailed explanations

Part1 :

Kernel part, I found rational quadratic kernel formula from website .

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

Use alpha=1,sigma=1,lengthscale=1 to calculate kernel in this part.
And I also import scipy.spatial.distance.cdist to calculate Euclidean distance.

First, load data and call GP() to do Gaussian process.
Then, in the Gaussian Process, I try to find variance and mean of our prediction of test data to draw plot. I use class lecture formula to find variance and mean.

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

(48)

Finally, to find 95% confidence interval, I mark (mean + 1.96 * var) and (mean - 1.96 *var).

Part2:

First, I use scipy.optimize.minimize to find optimal parameter(sigma, alpha ,lengthscale).
My objective function is refers to class lecture(loglikelihood).

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

(52)

Second, I try to find a good initial value.
test_set is all parameter choice. I use for-loop to find the best one.

This part is in main function.

```
alpha=1
beta=5
sigma=1
lengh=1


if __name__ == '__main__':

    x,y=Load()


    #part1
    x_test = np.linspace(-60.0, 60.0, 1000).reshape(-1, 1)
    GP(x,y,x_test)




    #part2
    optimal=100000
    opt_ini=[]
    test_set=[30,20,10,1,0.1,0.01,0.001,0.0001]

    for sigma_test in test_set:
        for alpha_test in test_set:
            for lengh_test in test_set:
                opt = minimize(Objective, [sigma_test,alpha_test,lengh_test], bounds=((1e-8, 1e6), (1e-8, 1e6), (1e-8, 1e6)), args=(x, y, beta))
                if opt.fun < optimal:
                    print('-')
                    opt_ini=[sigma_test,alpha_test,lengh_test]
                    alpha=opt.x[0]
                    sigma=opt.x[1]
                    lengh=opt.x[2]
                    optimal=opt.fun

    print(opt_ini)
    print(alpha,sigma,lengh)

    GP(x,y,x_test)
```

啟用 Windows

This part show how I do gaussian process.

```
def GP(x,y,x_test):


    kernel = RQK(x, x,alpha,sigma,lengh)
    kernel_star = RQK(x, x_test,alpha,sigma,lengh)
    kernel_star_star = RQK(x_test, x_test,alpha,sigma,lengh)


    C = kernel + np.identity(len(x), dtype=np.float64) * (1 / beta)
    C_inv = np.linalg.inv(C)

    mean = kernel_star.T @ C_inv @ y
    var = kernel_star_star + np.identity(len(x_test), dtype=np.float64) * (1 / beta) - kernel_star.T @ C_inv @ kernel_star
    var = 1.96*np.sqrt(np.diag(var))#95%信賴區間

    x_test = x_test.ravel()
    mean = mean.ravel()

    plt.plot(x_test, mean, color='b')
    plt.scatter(x, y, color='k', s=10)


    plt.plot(x_test, mean + var, color='r')
    plt.plot(x_test, mean - var, color='r')
    plt.fill_between(x_test, mean + var, mean - var, color='r', alpha=0.1)

    plt.xlim(-60, 60)
    plt.show()
```

This part show how I do rational quadratic kernel.

```
def RQK(X1, X2,alpha,sigma,lengh):
    kernel = (sigma ** 2) * ((cdist(X1, X2, 'sqeuclidean') / 2 * alpha * (lengh ** 2)) + 1) ** (-alpha)
    return kernel
```

This part show the objective function

```
def Objective(theta, X, Y, beta):
    theta = theta.ravel()
    kernel = RQK(X, X,theta[0],theta[1],theta[2]) + np.identity(len(X), dtype=np.float64) * (1 / beta)
    result = np.sum(np.log(np.diagonal(np.linalg.cholesky(kernel)))) + 0.5 * Y.T @ np.linalg.inv(kernel) @ Y + 0.5 * len(X) * np.log(2 * np.pi)
    return result
```

b. experiments settings and results

Part1

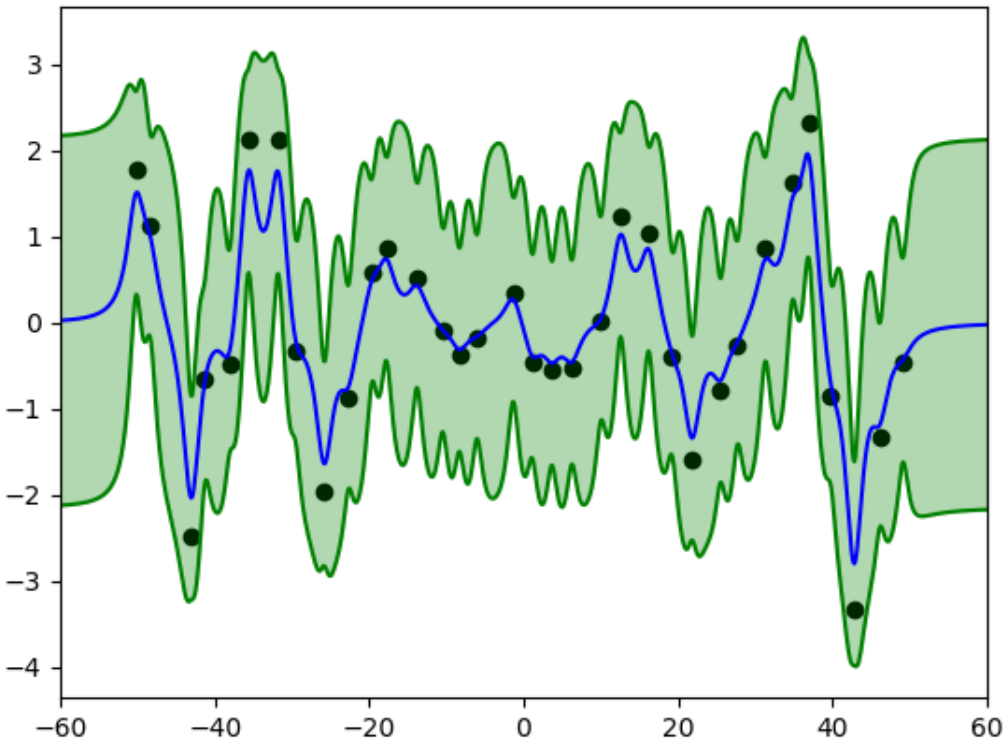
Parameter:

- Alpha : 1
- Sigma : 1
- Lengthscale : 1

Black scatter dot is all training data point.

Blue line represent the mean of f in range [-60,60]

Green part represent 95% confidence interval of f



Part2

I choose initial value from test_set=[30, 20, 10, 1, 0.1, 0.01, 0.001, 0.0001]
Finally, I find that using alpha=20 sigma=0.01 lengthscale=10 as initial value will have best performance.

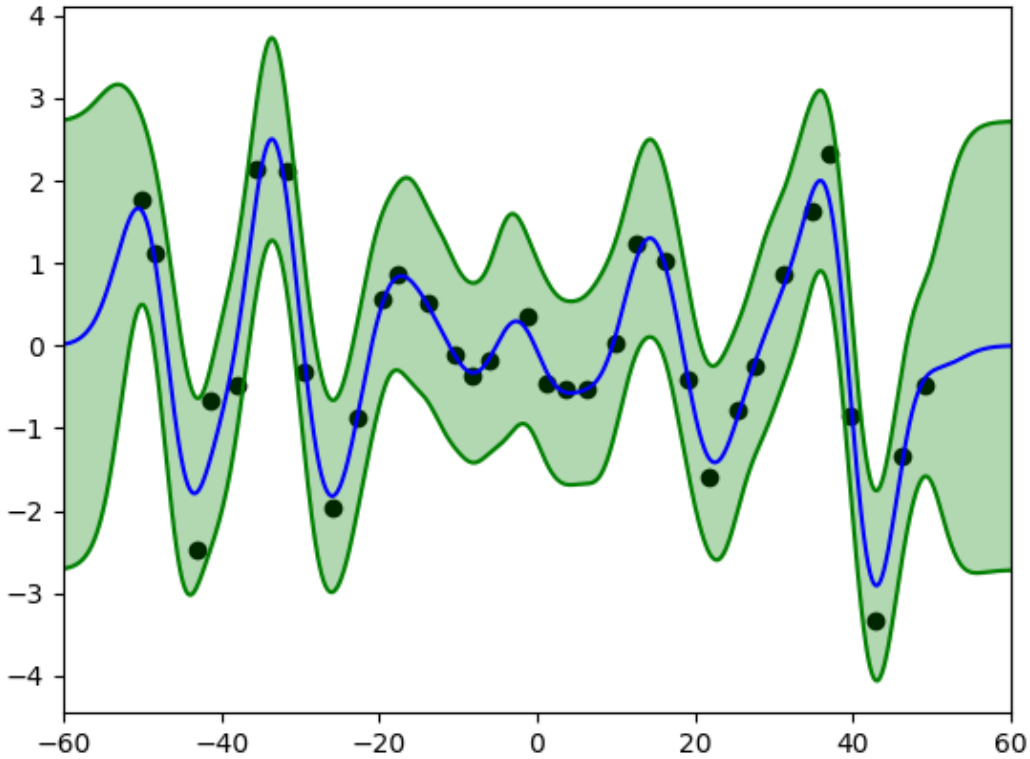
And Optimal kernel parameters will be alpha= 352.95533045415516, sigma= 1.3132274483474502 ,lengthscale= 0.0008541559815780655

Initial parmater:

alpha=20
sigma=0.01
lengthscale=10

After Optimize:

alpha= 352.95533045415516
sigma= 1.3132274483474502
lengthscale= 0.0008541559815780655



c. observations and discussion

After optimize the parameter, the result line become smooth . Variance becoming more smooth represent better performance because it mean we can predict ϵi more accuractely. (Yi = f(Xi) + ϵi)

In part 2, I found that how to decide test_set is a good question, because the speed will lower if we choose more test value.But if we choose too few value, its performance may be bad.

II. SVM

a. code with detailed explanations

Part1

I find svm_train parameter option from website, so I use “-t” to change kernel type. Use svm_train with train data to get a model, and then use svm_predict with test data and the model to check model performance.

From website:

```
-s svm_type : set type of SVM (default 0)
    0 -- C-SVC(multi-class classification)
    1 -- nu-SVC(multi-class classification)
    2 -- one-class SVM
    3 -- epsilon-SVR(regression)
    4 -- nu-SVR(regression)
-t kernel_type : set type of kernel function (default 2)
    0 -- linear: u'*v
    1 -- polynomial: (gamma*u'*v + coef0)^degree
    2 -- radial basis function: exp(-gamma*|u-v|^2)
    3 -- sigmoid: tanh(gamma*u'*v + coef0)
    4 -- precomputed kernel (kernel values in training_set_file)
-d degree : set degree in kernel function (default 3)
-g gamma : set gamma in kernel function (default 1/num_features)
-r coef0 : set coef0 in kernel function (default 0)
-c cost : set the parameter C of C-SVC, epsilon-SVR, and nu-SVR (default 1)
-n nu : set the parameter nu of nu-SVC, one-class SVM, and nu-SVR (default 0.5)
-p epsilon : set the epsilon in loss function of epsilon-SVR (default 0.1)
-m cachesize : set cache memory size in MB (default 100)
-e epsilon : set tolerance of termination criterion (default 0.001)
-h shrinking : whether to use the shrinking heuristics, 0 or 1 (default 1)
-b probability_estimates : whether to train a SVC or SVR model for probability estimates, 0 or 1 (default 0)
-wi weight : set the parameter C of class i to weight*C, for C-SVC (default 1)
-v n: n-fold cross validation mode
-q : quiet mode (no outputs)
```

This code show I do with linear kernel, polynomial kernel and RBF kernel.

```
if __name__ == '__main__':
    x_train , y_train , x_test , y_test = load_data()

    model=svm_train(y_train,x_train,'-q -t 0')
    p_label,p_acc,p_vals=svm_predict(y_test,x_test,model,'-q')
    linear_accuracy=p_acc[0]
    print("linear kernel accuracy : " + str(linear_accuracy)+"%")

    model=svm_train(y_train,x_train,'-q -t 1')
    p_label,p_acc,p_vals=svm_predict(y_test,x_test,model,'-q')
    poly_accuracy=p_acc[0]
    print("polynomial kernel accuracy : " + str(poly_accuracy)+"%")

    model=svm_train(y_train,x_train,'-q -t 2')
    p_label,p_acc,p_vals=svm_predict(y_test,x_test,model,'-q')
    rbf_accuracy=p_acc[0]
    print("radial basis function : " + str(rbf_accuracy)+"%")
```

Part2

Svm_train default SVM_type is CSVM, so we don't need to change this parameter option.

I set parameter "-g -c -v -t", -v mean it will do in cross-validation, -g mean gamma,-c mean cost, -t mean kernel type.

varable gamma [ ] and cost [ ] show all possible value, I use for-loop to compute all result (by svm_train()) and store in a np.array .And then find the best result we store in this array by using np.unravel_index(np.argmax(result)) .

code:

```
def grid(x_train , y_train , x_test , y_test):
    gamma=[0.001,0.01, 0.5, 0.1, 1,5,10,15]
    cost=[ 0.001,0.01,0.1, 1,5,10,15]

    result=np.zeros((3,len(gamma),len(cost)))

    for i_gamma in range(len(gamma)):
        for i_cost in range(len(cost)):
            for kernel_t in range(3):

                para='-q -v 3 -t '
                para+=str(kernel_t)
                para+=' -g '
                para+=str(gamma[i_gamma])
                para+=' -c '
                para+=str(cost[i_cost])
                print(para)
                acc=svm_train(y_train,x_train,para)
                result[kernel_t][i_gamma][i_cost]=acc
                print(acc)

    return result


if __name__ == '__main__':
    x_train , y_train , x_test , y_test = load_data()
    result = grid(x_train , y_train , x_test , y_test)
    print(result)
    ind = np.unravel_index(np.argmax(result, axis=None), result.shape)

    print(result[ind[0]][ind[1]][ind[2]])
```

Part3

I found how to use user-defined kernel in libsvm from website.
(https://blog.csdn.net/a200800170331/article/details/43486913)
First, I compute the new kernel(Linear+RBF).
And here I also import scipy.spatial.distance.cdist to calculate Euclidean distance(in RBF kernel).

Second, using svm_problem to let it fit svm_train parameter format.
Finally, use svm_train to find the model, and then using svm_predict test its performance.

Code:

```python
def new_kernel(x,y):
    gamma=1

    linear_k=x @ y.T
    RBF_k=np.exp(-gamma*cdist(x,y, 'sqeuclidean'))

    k=linear_k+RBF_k
    k=np.hstack((np.arange(1,len(x)+1).reshape(-1,1),k))

    return k



if __name__ == '__main__':
    x_train , y_train , x_test , y_test = load_data()


    kernel=new_kernel(x_train,x_train)

    prob=svm_problem(y_train,kernel,isKernel=True)


    model=svm_train(prob,'-t 4 -q')

    kernel_test=new_kernel(x_test, x_train)
    p_label,p_acc,p_vals=svm_predict(y_test,kernel_test,model,'-q')
    print('linear kernel + RBF kernel: {:.2f}%'.format(p_acc[0]))
```

b. experiments settings and results

Part1

I only change kernel type “-t”, and another parameter option use default value.

Accuracy:

Linear kernel : 95.08%

Polynomial kernel : 34.68%

RBF : 95.32%

Part2

possible value:

gamma=[0.001, 0.01, 0.5, 0.1, 1, 5, 10, 15]

cost =[0.001, 0.01, 0.1, 1, 5, 10, 15]

After Grid search, I found best parameter value is

Gamma = 0.01

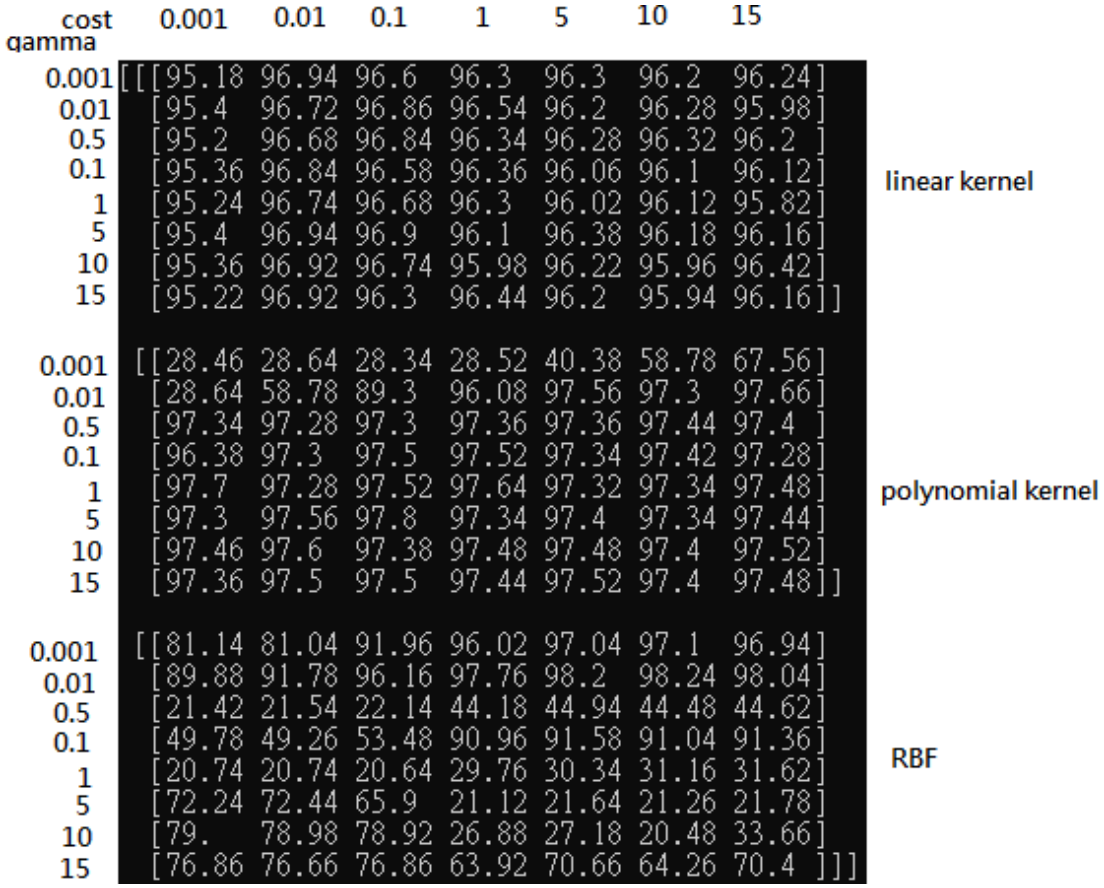
Cost = 10

Kernel = RBF

Cross Validation Accuracy:

Accuracy : 98.24%

Following ndarray record each Cross Validation Accuracy (accordind to each parameter value) :



Part3

I use kernel type “-t 4” and new kernel (linear kernel + RBF)
(other parameter use default)

Accuracy : 95.64%

Its performance seems to be between linear kernel and RBF kernel.

c. observations and discussion (10%)

I found that linear kernel is a stable kernel type. C-SVC with Linear kernel only need to search “cost” parameter, and many value of cost can get a good accuracy. I think it is a good advantage because grid search always take much time.

However, RBF has the best performance although it need to take time to find the best parameter (grid search).

When collecting information, I also found many kinds of kernels, and we can also combine these kernels (-t 4). Each kind of kernel has its own advantages, so how to choose the most suitable kernel for each problem is a good question.