HW05

資工所碩一 吳承翰 0856105

code: https://github.com/chiha8888/NCTU-ML-class/tree/master/HW05

1. Gaussian process

1-1:

Rational Quadratic Kernel alpha,length_scale parameters set to 1

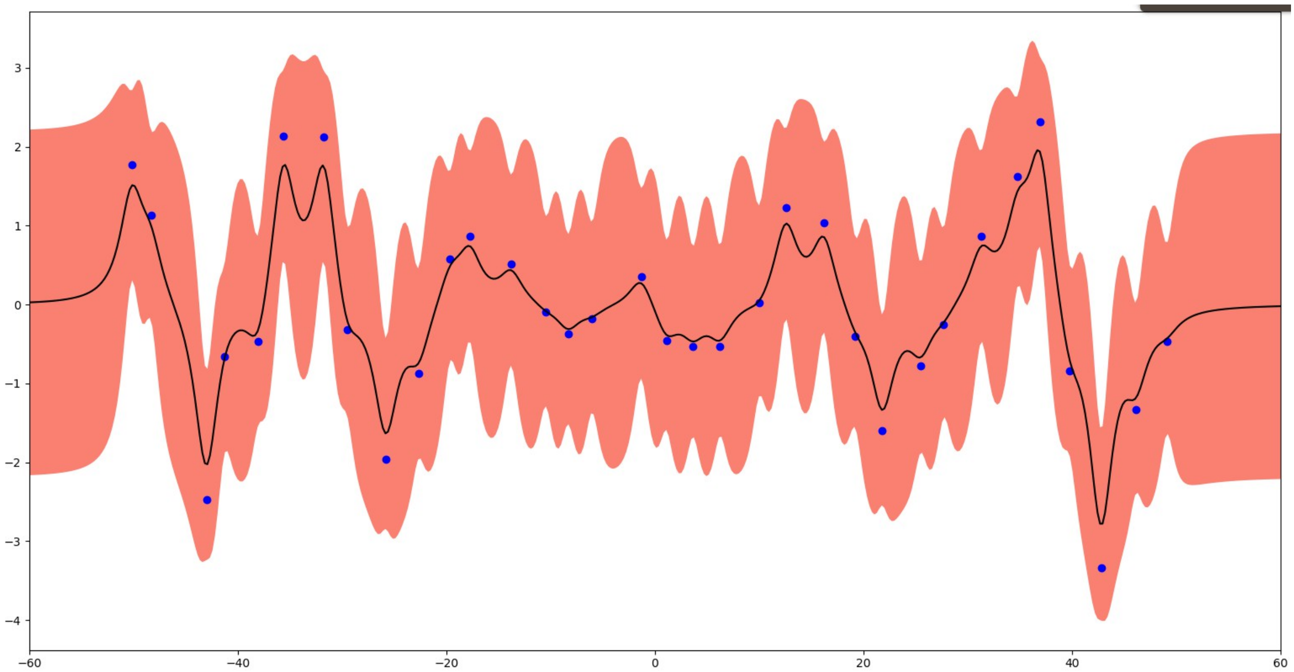
First using kernel() to calculate data points' similarity, then using predict() to find np.linspace(-60,60,num=500) sampling points' mean & variance.

```
X,y=load_data()
beta=5

#kernel
K=kernel(X,X,alpha=1,length_scale=1)+1/beta*np.identity(len(X))

# mean and variance in range[-60,60]
x_line=np.linspace(-60,60,num=500)
mean_predict,variance_predict=predict(x_line,X,y,K,beta,alpha=1,length_scale=1)
mean_predict=mean_predict.reshape(-1)
variance_predict=np.sqrt(np.diag(variance_predict))

#plot
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.show()
```

```
def kernel(X1,X2,alpha=1,length_scale=1):
    '''
    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
    '''
    square_error=np.power(X1.reshape(-1,1)-X2.reshape(1,-1),2.0)
    kernel=np.power(1+square_error/(2*alpha*length_scale**2),-alpha)

    return kernel
```

```
def predict(x_line,X,y,K,beta,alpha=1,length_scale=1):
    '''
    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=kernel(X,x_line,alpha=1,length_scale=1)
    k_xstar_xstar=kernel(x_line,x_line,alpha=1,length_scale=1)
    means=k_x_xstar.T @ np.linalg.inv(K) @ y.reshape(-1,1)
    vars=k_xstar_xstar+(1/beta)*np.identity(len(k_xstar_xstar))-k_x_xstar.T @ np.linalg.inv(K) @ k_x_xstar

    return means,vars
```

Result：



1-2:

using scipy.optimize minimize objective function

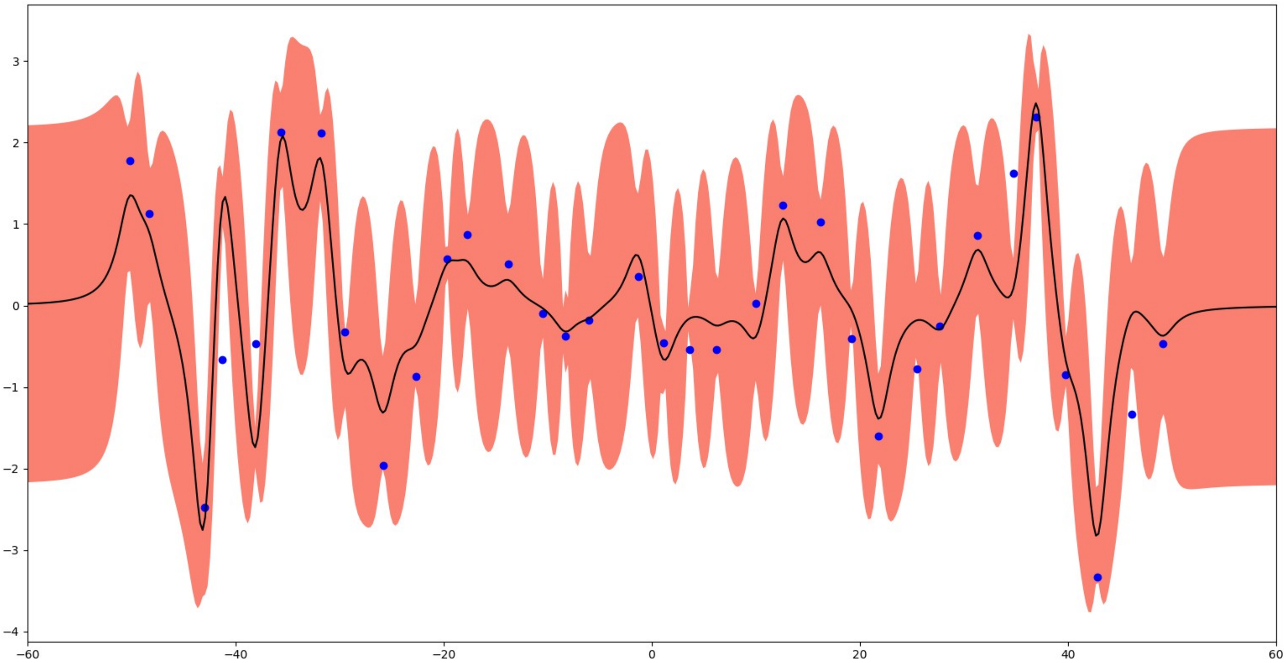
$$\log p(\mathbf{y}|\mathbf{X}) = -\frac{1}{2}\mathbf{y}^T\boldsymbol{\alpha} - \sum_i \log L_{ii} - \frac{N}{2}\log(2\pi)$$

Finding optimal alpha & optimal length_scale to minimize negative logp(y|X)，respectively using [0.01, 0.1, 0, 1, 10, 100] as initial value to run minimize().

```
X,y=load_data()
beta=5

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_value = res.fun
            alpha_optimize,length_scale_optimize=res.x
print('alpha: ',alpha_optimize)
print('length_scale: ',length_scale_optimize)

#kernel
K=kernel(X,X,alpha=alpha_optimize,length_scale=length_scale_optimize)+1/beta*np.identity(len(X))

# mean and variance in range[-60,60]
x_line=np.linspace(-60,60,num=500)
mean_predict,variance_predict=predict(x_line,X,y,K,beta,alpha=alpha_optimize,length_scale=length_scale_optimize)
mean_predict=mean_predict.reshape(-1)
variance_predict=np.sqrt(np.abs(np.diag(variance_predict)))

#plot
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')
```

```
def objective_function(X,y,beta):
    '''
    :param X:  (n) ndarray
    :param y:  (n) ndarray
    :param beta:
    :return:
    '''
    def objective(theta):
        K=kernel(X,X,alpha=theta[0],length_scale=theta[1])+(1/beta)*np.identity(len(X))
        L=np.linalg.cholesky(K)
        return 0.5*y.reshape(1,-1) @ np.linalg.inv(K) @ y.reshape(-1,1) + np.sum(np.log(np.diag(L))) + 0.5*len(X)*np.log(2*np.pi)

    return objective
```

Result：
comparing with alpha=length_scale=1，the variance at the data point is much smaller.



2.SVM on MNIST
2-1：

```
if __name__=='__main__':
    X_train=load_x('X_train.csv')
    y_train=load_y('Y_train.csv')
    X_test=load_x('X_test.csv')
    y_test=load_y('Y_test.csv')
    kernel_types={'linear':'-t 0','polynomial':'-t 1','radial basis function':'-t 2'}

    accuracy=[]
    for k,param in kernel_types.items():
        model=svm_train(y_train,X_train,'-q '+param)
        p_label,p_acc,p_vals=svm_predict(y_test,X_test,model,'-q')
        accuracy.append(p_acc[0])

    i=0
    for k,v in kernel_types.items():
        print('{} kernel accuracy: {:.2f}%'.format(k,accuracy[i]))
        i+=1
```

all using default parameters

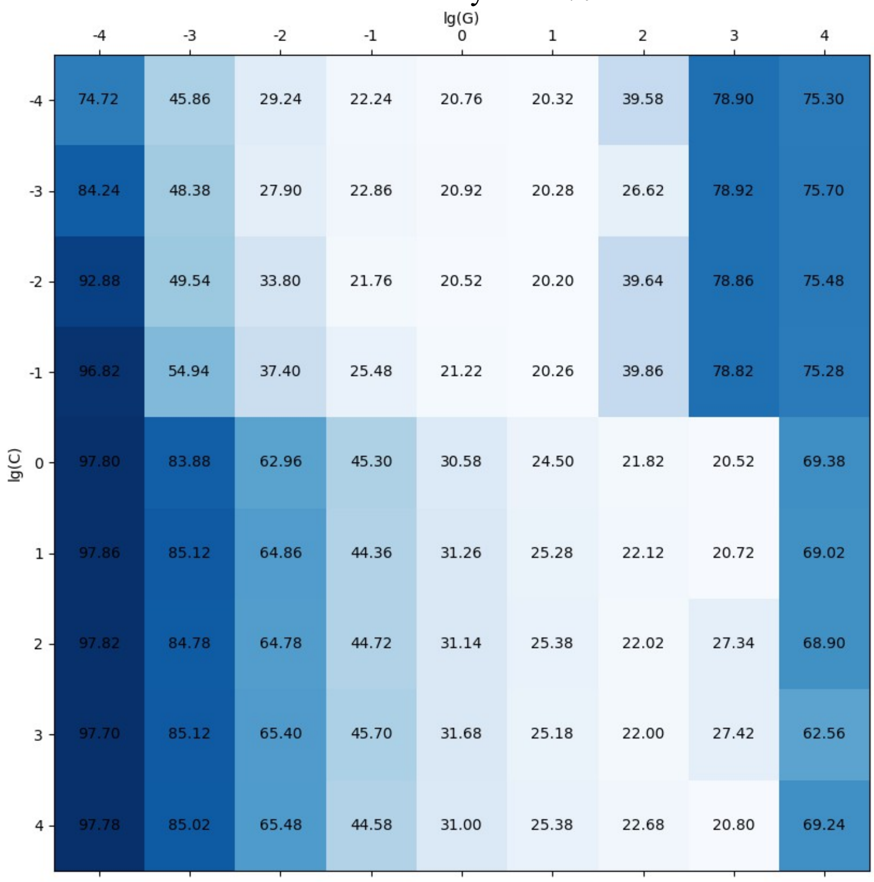
Result：
linear kernel accuracy: 95.08%
polynomial kernel accuracy: 34.68%
radial basis function kernel accuracy: 95.32%

2-2 :
using C and gamma from 2^-4~2^4 to do grid search. If -v is specified, cross validation is conducted and the return of svm_train() is just a scalar.

```python
if __name__=='__main__':
    X_train=load_x('X_train.csv')
    y_train=load_y('Y_train.csv')
    X_test=load_x('X_test.csv')
    y_test=load_y('Y_test.csv')

    log2c=log2g=[-4,-3,-2,-1,0,1,2,3,4]
    confusion_matrix=grid_search(log2c,log2g,X_train,y_train,X_test,y_test)

    plot_confusion_matrix(confusion_matrix,log2c,log2g)
```

```python
def grid_search(log2c,log2g,X_train,y_train,X_test,y_test):
    confusion_matrix=np.zeros((len(log2c),len(log2g)))
    for i in range(len(log2c)):
        for j in range(len(log2g)):
            param='-q -t 2 -v 3 -c {} -g {}'.format(2**log2c[i],2**log2g[j])
            acc=svm_train(y_train,X_train,param)
            confusion_matrix[i,j]=acc
    return confusion_matrix
```

Result:
C=2^4 , gamma=2^-4 has the best validation accuracy 97.78%



2-3:
using Linear kernel + RBF kernel to get a new kernel, gamma parameter set 2^-4 from 2-2 (which has better predicted result), scipy.spacial.distance.pdist is to calculate the pairwise distance between two data points.

```
def precomputed_kernel(X,gamma):
    kernel_linear=X @ X.T
    kernel_RBF=squareform(np.exp(-gamma*pdist(X,'sqeuclidean')))
    kernel=kernel_linear+kernel_RBF
    kernel=np.hstack((np.arange(1,len(X)+1).reshape(-1,1),kernel))
    return kernel

if __name__=='__main__':
    X_train=load_x('X_train.csv')
    y_train=load_y('Y_train.csv')
    X_test=load_x('X_test.csv')
    y_test=load_y('Y_test.csv')


    kernel_train=precomputed_kernel(X_train,2**-4)
    prob=svm_problem(y_train,kernel_train,isKernel=True)
    param=svm_parameter('-q -t 4')
    model=svm_train(prob,param)

    kernel_test=precomputed_kernel(X_test,2**-4)
    p_label,p_acc,p_vals=svm_predict(y_test,kernel_test,model,'-q')
    print('linear kernel + RBF kernel accuracy: {:.2f}%'.format(p_acc[0]))
```

Result:
linear kernel + RBF kernel accuracy: 23.40%