Hw5 Report

0856041 范嘉容

1. Gaussian Process

- Process input data, use 120 points for testing, initial kernel parameter: (sigma=1.0, alpha=1.0, lengthscale=1.0)

```
train_x, train_y = read_input()
test_x = np.linspace(-60, 60, 120).reshape(-1,1)
k_params = [1.0, 1.0, 1.0] # initial kernel parameters
```

- Use rational quadratic kernel

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

With:

- σ^2 the overall variance
- ℓ the lengthscale
- α the scale-mixture ($\alpha > 0$)

- Training

$$p(\mathbf{y}) \sim N(\mathbf{y}|\mathbf{0}, \mathbf{C})$$
$$\mathbf{C}(\mathbf{x}_n, \mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m) + \beta^{-1}\delta_{nm}$$

```
Beta = 5
I = np.identity(train_x.shape[0])
C = kernel(train_x, train_x, k_params) + 1/Beta*I
```

- Prediction
  1. Calculate kernel
     (data points' similarity)

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

  2. Calculate conditional distribution
     (mean and covariance)

$$y^* = f(\mathbf{x}^*)$$
$$p(y^*|\mathbf{y}) \sim N(\mu(\mathbf{x}^*), \sigma^2(\mathbf{x}^*))$$
$$\mu(\mathbf{x}^*) = k(\mathbf{x}, \mathbf{x}^*)^T \mathbf{C}^{-1} \mathbf{y}$$
$$\sigma^2(\mathbf{x}^*) = k^* - k(\mathbf{x}, \mathbf{x}^*)^T \mathbf{C}^{-1} k(\mathbf{x}, \mathbf{x}^*)$$

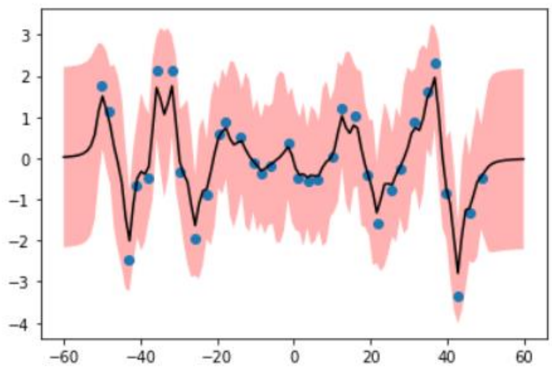
```
    kernel_train_test = kernel(train_x, test_x, k_params)
    kernel_test_test = kernel(test_x, test_x, k_params)

    C_inv = np.linalg.inv(C)
    mu = (kernel_train_test.T)@C_inv@train_y
    cov = kernel_test_test+1/Beta - (kernel_train_test.T)@C_inv@kernel_train_test
```

- Visualization

  Blue dot: training data points

  Black line: mean of f

  Red region: 95% confidence interval of f



- Optimize the kernel parameters

  Covariance function:

$$\mathbf{C}_\theta = k_\theta(\mathbf{x}_n, \mathbf{x}_m)$$

Marginal likelihood

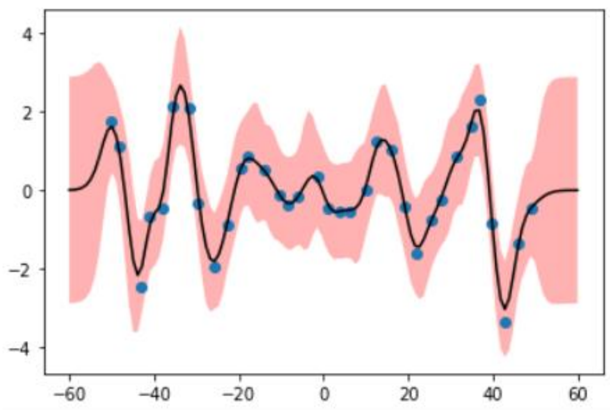
$$p(\mathbf{y}|\theta) \sim N(\mathbf{y}|\mathbf{0}, \mathbf{C}_\theta)$$

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

```
def nmll(k_params, train_x, train_y):
    '''
    negative marginal log-likelihood
    '''
    C = kernel(train_x, train_x, k_params)
    term1 = 0.5 * np.log(np.linalg.det(C))
    term2 = 0.5 * (train_y.T)@np.linalg.inv(C)@train_y
    term3 = (train_x.shape[0]/2.0) * np.log(2*np.pi)
    return (term1 + term2 + term3).item()
```

```
    fmin = minimize(fun=nmll, x0=k_params, args=(train_x, train_y))
```

```
Optimized kerenel parameters:
sigma=1.372307, alpha=8.490846, lengthscale=2.476118
```



Compare with the figure with initial kernel parameters, in this figure: the red region (variance) is smaller.

2. SVM

2-1.

```python
def svm_diff_kernel(train_x, train_y, test_x, test_y):

    print('Linear kernel function:')
    problem = svm_problem(train_y, train_x)
    parameter = svm_parameter('-q -t 0')
    model = svm_train(problem, parameter)
    svm_predict(test_y, test_x, model)

    print('Polynomial kernel function:')
    problem = svm_problem(train_y, train_x)
    parameter = svm_parameter('-q -t 1')
    model = svm_train(problem, parameter)
    svm_predict(test_y, test_x, model)

    print('RBF kernel function:')
    problem = svm_problem(train_y, train_x)
    parameter = svm_parameter('-q -t 2')
    model = svm_train(problem, parameter)
    svm_predict(test_y, test_x, model)
```

Result:

```
Linear kernel function:
Accuracy = 95.08% (2377/2500) (classification)
Polynomial kernel function:
Accuracy = 34.68% (867/2500) (classification)
RBF kernel function:
Accuracy = 95.32% (2383/2500) (classification)
```

2-2.

Grid search:

c: from 2^-4 to 2^4

gamma: from 2^-4 to 2^4

```python
def grid_search(train_x, train_y):
    C = [2**-4, 2**-3, 2**-2, 2**-1, 1, 2**1, 2**2, 2**3, 2**4]
    gamma = [2**-4, 2**-3, 2**-2, 2**-1, 1, 2**1, 2**2, 2**3, 2**4]

    #C = [1/16]
    #gamma = [1/16]
    acc_matrix = np.zeros((len(C),len(gamma)))
    best_cg = (0,0)
    best_acc = 0

    for c in C:
        for g in gamma:
            print('C=%f, g=%f' % (c,g))
            problem = svm_problem(train_y, train_x)
            parameter = svm_parameter('-q -s 0 -t 2 -v 3 -c {} -g {}'.format(c,g))
            acc = svm_train(problem, parameter)

            fillin_acc_matrix(c,g,acc,acc_matrix)
            if acc > best_acc:
                best_cg = (c,g)
                best_acc = acc
        print(acc_matrix)

    return best_cg, best_acc
```

Result:

When c=2^4, gamma=2^-4, reach best accuracy:

Cross validation accuracy = 97.92%

2-3.

Kernel: linear kernel + RBF kernel

c = 2^4, gamma=2^-4 (from 2-2)

use scipy.distance to calculate the distance between points

```python
def user_defined_kernel(train_x, test_x):
    gamma = 1/16

    train_linear = train_x@(train_x.T)
    train_rbf = squareform(np.exp(-gamma * pdist(train_x, 'sqeuclidean')))
    train_kernel = np.hstack((np.arange(1,train_x.shape[0]+1).reshape((-1,1)), train_linear+train_rbf))

    test_linear = test_x@(train_x.T)
    test_rbf = np.exp(-gamma * cdist(test_x, train_x, 'sqeuclidean'))
    test_kernel = np.hstack((np.arange(1,test_x.shape[0]+1).reshape((-1,1)), test_linear+test_rbf))

    return train_kernel, test_kernel
```

```python
def svm_user_defined_kernel(train_x, train_y, test_x, test_y):
    problem = svm_problem(train_y, train_x, isKernel=True)
    parameter = svm_parameter('-q -s 0 -t 4 -c 16 -g 0.0625')
    model = svm_train(problem, parameter)
    svm_predict(test_y, test_x, model)
```

Result:

```
Accuracy = 26.4% (660/2500) (classification)
```