0880803 – 馮清海

Thanh – Hai, Phung

HOMEWORK 5

MACHINE LEARNING

1. Gaussian Process

Reading input

<code>

```
x = []
y = []
with open('input.data') as file:
    for line in file:
        x.append(float(line.split()[0]))
        y.append(float(line.split()[1]))
train_x = np.array(x).reshape(-1, 1)
train_y = np.array(y).reshape(-1, 1)

beta = 5
noise = 1 / beta
test_x = np.arange(-60, 60, 1).reshape(-1, 1)
```

Define quadratic kernel

<code>

```
"""
Define kernel
"""

def kernel(x1, x2, length=1.0, sigma=1.0, alpha=1.0):
    return (sigma ** 2) * (1 + cdist(x1, x2, 'sqeuclidean') / (2 * alpha * length ** 2)) ** (-alpha)
```

Define Posterior Gaussian Process

<code>

```python
"""
Gaussian Process Predict
"""

def GP_predict(test_x, train_x, train_y, length=1.0, sigma=1.0, alpha=1.0, noise=1 / 5):
    k = kernel(train_x, train_x, length, sigma, alpha) + noise * np.eye(len(train_x))
    k_train = kernel(train_x, test_x, length, sigma, alpha)
    mean_test_y = kernel(test_x, test_x, length, sigma, alpha) + noise
    k_inv = np.linalg.inv(k)
    mean = np.linalg.multi_dot([k_train.T, k_inv, train_y])
    covariance = mean_test_y - np.linalg.multi_dot([k_train.T, k_inv, k_train])
    return mean, covariance
```

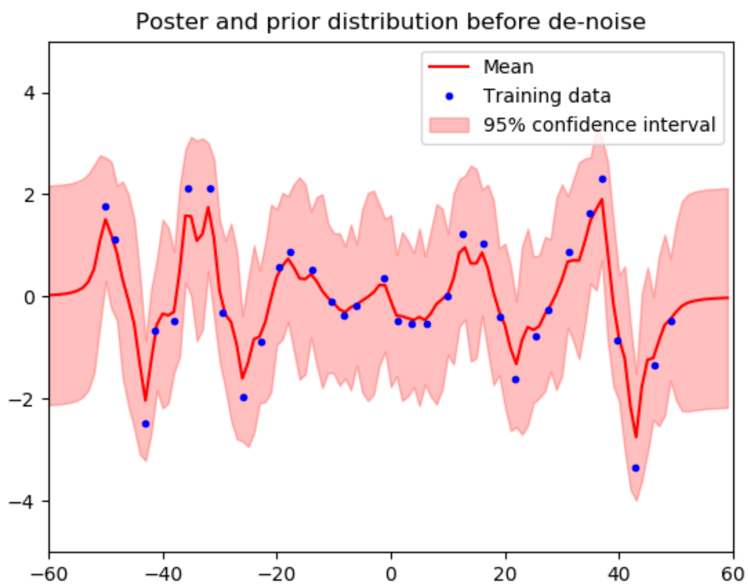
Define Plot function

<code>

```python
"""
Gaussian Process Predict Plot
"""
def GP_plot(mean, covariance, test_x, train_x, train_y, fig_name):
    test_x = test_x.ravel()
    mean = mean.ravel()
    temp = 1.96 * np.sqrt(np.diag(covariance))
    plt.figure()
    plt.fill_between(test_x, mean + temp, mean - temp, alpha=0.25, color='r', label='95% confidence interval')
    plt.plot(test_x, mean, label='Mean', color='r')
    plt.plot(train_x, train_y, 'b.', label='Training data')
    plt.axis([-60, 60, -5, 5])
    plt.legend()
    plt.title(fig_name)
    plt.savefig(fig_name +'.png')
    # plt.show()
```

Before de-noise

```python
"""
Before de-noise
"""
GPdefault = True
if GPdefault == True:
    mean, covar = GP_predict(test_x, train_x, train_y, noise=noise)
    GP_plot(mean, covar, test_x, train_x=train_x, train_y=train_y,
            fig_name='Poster and prior distribution before de-noise')
```

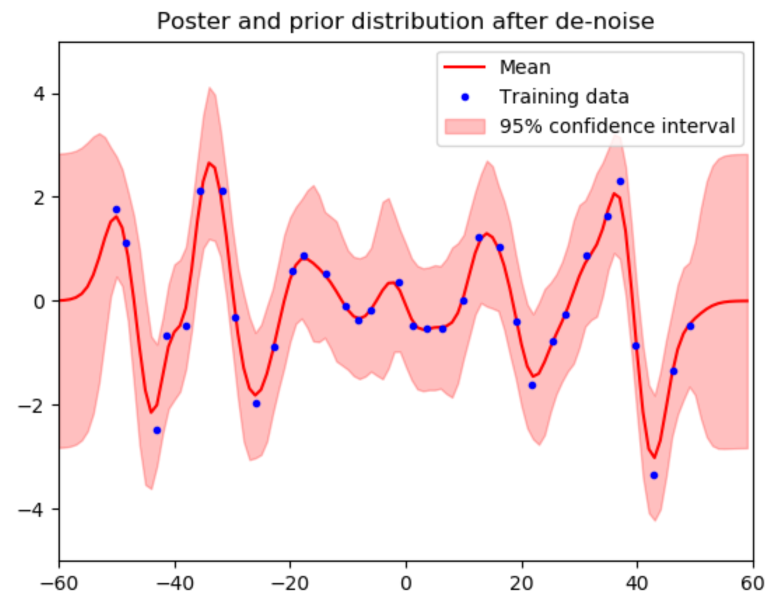


After de-noise

<code>

```python
"""
After de-noise
"""
GPoptimize = True
if GPoptimize == True:
    def NLL(test_x, train_x, train_y):
        length = test_x[0]
        sigma = test_x[1]
        alpha = test_x[2]
        k = kernel(train_x, train_x, length, sigma, alpha)
        temp1 = 0.5 * np.log(np.linalg.det(k))
        temp2 = 0.5 * np.linalg.multi_dot([train_y.T, np.linalg.inv(k), train_y])
        temp3 = 0.5 * train_x.shape[0] * np.log(2 * np.pi)
        return temp1 + temp2 + temp3

    optimize_result = minimize(fun=NLL, x0=np.array([1, 1, 1]), args=(train_x, train_y),
                               bounds=((1e-3, None), (1e-3, None), (1e-3, None)), method='L-BFGS-B')
    print(optimize_result.x)
    length_op, sigma_op, alpha_op = optimize_result.x
    mean, covar = GP_predict(test_x, train_x=train_x, train_y=train_y, length=length_op, sigma=sigma_op, alpha=alpha_op,
                             noise=noise)
    GP_plot(mean, covar, test_x, train_x=train_x, train_y=train_y,
            fig_name='Poster and prior distribution after de-noise')
```



II. Support Vector Machine (SVM)

Reading input

```
"""
Read input
"""

with open('X_train.csv') as file:
    csv_reader = csv.reader(file, delimiter=',')
    x_train = list(csv_reader)
    x_train = [[float(y) for y in x] for x in x_train]

with open('Y_train.csv') as file:
    csv_reader = csv.reader(file, delimiter=',')
    y_train_2nd = list(csv_reader)
    y_train = [y for x in y_train_2nd for y in x]
    y_train = [int(x) for x in y_train]

with open('X_test.csv') as file:
    csv_reader = csv.reader(file, delimiter=',')
    x_test = list(csv_reader)
    x_test = [[float(y) for y in x] for x in x_test]

with open('Y_test.csv') as file:
    csv_reader = csv.reader(file, delimiter=',')
    y_test_2nd = list(csv_reader)
    y_test = [y for x in y_test_2nd for y in x]
    y_test = [int(x) for x in y_test]
```

<code>

1. Use difference kernel function (by build-in kernels) (No Cross – Validation)

<code>

```
"""
Compare the different between three kernel function
"""
Comparemode = False
if Comparemode == True:
    kernel = ['Linear Kernel', 'Polynomial Kernel', 'RBF Kernel']
    for i in range(len(kernel)):
        print('Kernel Function: {}'.format(kernel[i]))
        prob = svm_problem(y_train, x_train)
        param = svm_parameter('-t {} -q'.format(i))
        model = svm_train(prob, param)
        model_predict = svm_predict(y_test, x_test, model)
```

The setting default built in run thru three kernel functions and run in test set.

Table 1: Comparison of 3 different default kernels

Kernel	Setting	Accuracy
Linear	c = 1	**95.08%** (2377/2500)
Polynomial	c = 1, d = 3, gamma = 0, r = 0	**34.68%** (867/2500)
Radial Basis Function	gamma = 0	**95.32%** (2382/2500)

2. C-SVC, Grid – Search, Cross – Validation

a. Linear, 3 – folds cross – validation, searching for variable c

<code>

```
#Linear kernel with 3 - fold cross validation
Linearmode = True
if Linearmode == True:
    cost = - 1
    gamma = -1
    degree = -1
    logset = []
    gridsearch = -1
    for log2c in range(-8, 1, 1):
        param = svm_parameter('-q -t 0 -v 3 -c {}'.format(2 ** log2c))
        prob = svm_problem(y_train, x_train)
        model = svm_train(prob, param)
        logset.append([2 ** log2c, model])
        if model > gridsearch:
            gridsearch = model
            cost = 2 ** log2c
    for log in logset:
        print(log)
    # Test accuracy
    param = svm_parameter('-q -t 0 -c {}'.format(cost))
    prob = svm_problem(y_train, x_train)
    model = svm_train(prob, param)
    model_predict = svm_predict(y_test, x_test, model)
```

Result of finding parameter c, performance on cross – validation

0.00390625	0.0078125	0.015625	**0.03125**	0.0625	0.125	0.25	0.5	1
96.3	96.82	96.899	**97.1**	97.08	96.76	96.32	96.06	96.24

- **c = 0.03125** gave the best performance on cross – validation
- Accuracy = **96%** (2400/2500) (classification)
- The accuracy is slightly higher than default setting
- The range of c is based on the suggestion of **libsvm** library. We firstly do the spare search to find a range of c, then make a fine-turn to get the better result

b. Polynomial, 3 – folds cross – validation, searching for variable c, gamma and degree

<code>

```python
# Polynomial kernel with 3 - fold cross validation
Polynomialmode = True
if Polynomialmode == True:
    cost = - 1
    gamma = -1
    degree = -1
    logset = []
    gridsearch = -1
    for d in range(1, 5, 1):
        for log2g in range(-2, 2, 1):
            for log2c in range(-8, 1, 1):
                param = svm_parameter('-q -t 1 -v 3 -c {} -g {} -d {}'.format(2 ** log2c, 2 ** log2g, d))
                prob = svm_problem(y_train, x_train)
                model = svm_train(prob, param)
                logset.append([2 ** log2c, 2 ** log2g, model])
                if model > gridsearch:
                    gridsearch = model
                    cost = 2 ** log2c
                    d = d
                    gamma = 2 ** log2g
    for log in logset:
        print(log)
    # Test accuracy
    param = svm_parameter('-q -t 1 -c {} -g {} -d {}'.format(cost, gamma, degree))
    prob = svm_problem(y_train, x_train)
    model = svm_train(prob, param)
    model_predict = svm_predict(y_test, x_test, model)
```

Result of finding parameter c, gamma and degree performance on cross – validation

c \ g	0.25	0.5	1	2
0.00390625	97.74	97.78	97.88	97.84
0.0078125	97.72	97.8	97.86	98.08
0.015625	98.04	97.98	97.88	97.88
0.03125	97.82	97.82	97.94	97.78
0.0625	**98.06**	97.619	97.96	97.619
0.125	97.82	97.92	97.89	98.0
0.25	97.92	97.94	97.88	97.98
0.5	97.76	97.84	98.02	97.8
1	97.86	97.72	97.89	97.94

- **c = 0.0125 and g = 0.25** gave the best performance on cross – validation
- Accuracy = **97.68%** (2442/2500) (classification)
- The accuracy is slightly higher than linear kernel
- The polynomial kernel required more time and memory for training compare to the linear kernel.

c. RBF kernel with 3 – folds cross validation, searching for variable c and gamma

<code>

```
# RBF kernel with 3 - fold cross validation
RBFmodel = True
if RBFmodel == True:
    cost = -1
    gamma = -1
    degree = -1
    param_label = []
    gridsearch = -1

    for log2g in range(-5, 2, 1):
        for log2c in range(-3, 9, 1):
            param = svm_parameter('-q -t 2 -v 3 -c {} -g {}'.format(2 ** log2c, 2 ** log2g))
            prob = svm_problem(y_train, x_train)
            model = svm_train(prob, param)
            param_label.append([2 ** log2c, 2 ** log2g, model])
            if model > gridsearch:
                gridsearch = model
                cost = 2 ** log2c
                gamma = 2 ** log2g
    for label in param_label:
        print(label)
    # Test accuracy
    param = svm_parameter('-q -t 2 -c {} -g {}'.format(cost, gamma))
    prob = svm_problem(y_train, x_train)
    model = svm_train(prob, param)
```

Result of finding parameter c and gamma, performance on cross – validation

c \ g	0.03125	0.0625	0.125	0.25	0.5	1	2
0.125	97.04	84.6	48.19	27.6	21.64	20.56	20.28
0.25	97.58	92.9	48.96	35.74	21.68	20.84	20.26
0.5	97.94	96.8	55.059	39.04	25.28	20.76	20.38
1	98.36	97.84	84.119	62.539	44.879	30.38	24.14

2	98.48	97.98	84.96	65.539	45.879	32.58	25.28
4	98.5	97.84	85.119	65.48	45.18	32.36	25.2
8	98.52	97.86	85.02	65.38	45.6	31.319	25.18
16	98.42	97.8	85.42	65.259	45.94	31.9	25.58
32	98.52	97.86	85.36	65.039	44.94	31.879	25.7
64	**98.56**	97.78	85.04	65.96	45.1	32.42	25.52
128	98.34	97.78	85.26	65.8	44.26	31.259	25.6
256	98.44	97.84	85.3	65.94	44.48	32.1	24.98

- **c = 64 and g = 0.03125** gave the best performance on cross – validation
- Accuracy = **98.52%** (2463/2500) (classification)
- The accuracy is slightly higher than linear kernel and polynomial kernel
- The polynomial kernel required more time and memory for training compare to the linear kernel and polynomial kernel.
- Since we may do pre-compute kernel, training time may be reduced.

3. Combine linear and RBF kernel

```
"""
Merger Linear and RBF kernel
"""
Mergemode = True
if Mergemode == True:
    #Compute kernel

    #Train
    negative_gamma = -1 / 4
    train_linear_kernel = np.matmul(x_train, np.transpose(x_train))
    train_rbf_kernel = squareform(np.exp(negative_gamma * pdist(x_train, 'sqeuclidean')))
    x_train_kernel = np.hstack((np.arange(1, 5001).reshape((5000, 1)), np.add(train_linear_kernel, train_rbf_kernel)))

    #Test
    test_linear_kernel = np.matmul(x_test, np.transpose(x_train))
    test_rbf_kernel = np.exp(negative_gamma * cdist(x_test, x_train, 'sqeuclidean'))
    x_test_kernel = np.hstack((np.arange(1, 2501).reshape((2500, 1)), np.add(test_linear_kernel, test_rbf_kernel)))

    #Predict new kernel
    prob = svm_problem(y_train, x_train_kernel, isKernel=True)
    param = svm_parameter('-t 4 -q')
    model = svm_train(prob, param)
    model_predict = svm_predict(y_test, x_test_kernel, model)
```

- Accuracy = **95.08%** (2377/2500) (classification)
- Compare with the previous kernel method

Method	Linear	Polynomial	RBF	Linear + RBF
Accuracy	96%	97.68%	98.52%	95.08%

- Using pre-compute kernel give the result faster than using build – in kernel functions when we do the grid search
- The RBF kernel has better accuracy than other method but slower in compute than other method, problem is also going with Polynomial compare to Linear Kernel. This can be trade – off for computation efficiency and accuracy.

If you have any concern about my code or other question don't hesitate to contact me!

My email is: *haiphung106@gmail.com*