ML_HW5_0856133_莊凱鈞

hw5q1

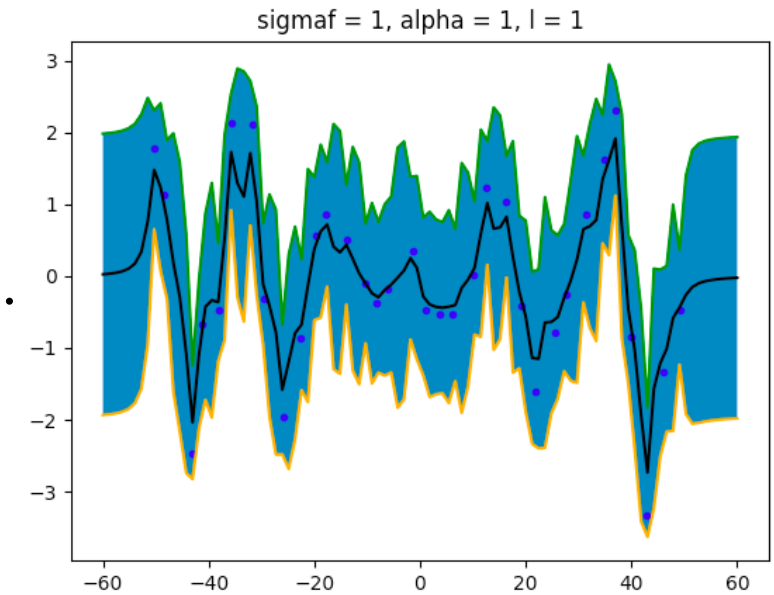
hw5q1_1: Apply Gaussian Process Regression to predict the distribution of f and visualize the result. Please use rational quadratic kernel to compute similarities between different points.

- gaussian process: given a domain, the corresponding continuous codomain is a random variable, and the joint of these random variables is gaussian process.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize

def rational_quadratic_kernel(xa, xb, sigmaf = 1, alpha = 1, l = 1):
    # sigmaf = 1.31
    # alpha = 423.11
    # l = 3.31
    numerator = np.abs(xa-xb)*np.abs(xa-xb)
    denominator = 2*alpha*l*l
    return sigmaf*sigmaf*np.power(1+numerator/denominator, -alpha)

def generate_x_star():
    x_star = np.random.uniform(-60, 60, 1)
    return x_star

def compute_K(X, Y, sigmaf=1, alpha=1, l=1):
    K = np.zeros((len(X), len(X)))
    for i in range(len(X)):
        for j in range(len(X)):
            K[i][j] = rational_quadratic_kernel(X[i], X[j], sigmaf, alpha, l)
            if i == j:
                K[i][j] += 0.2
    return K

def compute_K_star(X, Y, x_star, sigmaf=1, alpha=1, l=1):
    K_star = np.zeros(len(X))
    for i in range(34):
        K_star[i] = rational_quadratic_kernel(x_star, X[i], sigmaf, alpha, l)
    return K_star

def GPR():
    # load file
    X = np.zeros(34)
    Y = np.zeros(34)
    i = 0
    input_file = open("./input.data", "r")
    for line in input_file.readlines():
        X[i], Y[i] = line.split(' ')
        print(X[i], Y[i])
        i += 1

    # kernel
    sample_count = 100
    K = compute_K(X, Y)
    X_lin = np.linspace(-60, 60, num=sample_count)
    Y_lin = np.zeros(sample_count)
    var_y_star = np.zeros(sample_count)
    interval_upper = np.zeros(sample_count)
    interval_lower = np.zeros(sample_count)
    for i in range(len(X_lin)):
        K_star = compute_K_star(X, Y, X_lin[i])
        K_star_star = rational_quadratic_kernel(X_lin[i], X_lin[i])
        y_star_bar = K_star@np.linalg.inv(K)@Y
        Y_lin[i] = y_star_bar
        var_y_star[i] = K_star_star - K_star@np.linalg.inv(K)@np.transpose(K_sta
    for i in range(sample_count):
        interval_upper[i] = Y_lin[i] + 1.96*np.sqrt(var_y_star[i])
        interval_lower[i] = Y_lin[i] - 1.96*np.sqrt(var_y_star[i])

    # visualization
    plt.title("sigmaf = 1, alpha = 1, l = 1")
    plt.plot(X, Y, 'b.')
    plt.plot(X_lin, Y_lin, 'black')
    plt.fill_between(X_lin, interval_upper, interval_lower)
    plt.plot(X_lin, Y_lin + 1.96*np.sqrt(var_y_star), 'green')
    plt.plot(X_lin, Y_lin - 1.96*np.sqrt(var_y_star), 'orange')
    plt.show()
if __name__ == '__main__':
    GPR()
```

- rational quadratic kernel
  - $\sigma_f^2(1 + \frac{||x_a - x_b||^2}{2\alpha l^2})^{-\alpha}$
  - input: xa, xb

    - sigmaf, alpha, l are the kernel parameters that we want to optimize
- generate_x_star() is a function applied to uniformly generate data point between [-60, 60]
- compute_K() is a function applied to generate kernel matrix K
- compute_K_star() is a function applied to compute kernel between newly generated data point and the existing data point
- GPR() is the main program of our gaussian process regression
    - Step
        - we first load file and then do some preprocessing
        - we compute matrix K (34*34)
        - uniformly generate 100 data points between [-60, 60]
        - for each newly generated 100 data points, we compute K_star = [k(x*,x1), k(x*,x2), ..., k(x*, $x_{len(X)}$)], indicating the relation between x* and X in a higher dimensional space; we also compute K_star_star = k(x*, x*)
        - y_star_bar = K_star@np.linalg.inv(K)@Y
            - the formula comes from professor chiu's pdf
            - y_star_bar_i means the mean of the function for a given x_i
                - it could be thought of as passing x_i into the 34 functions, we'll get 34 values, and then find the mean of the 34 values
        - I compute the confidence interval, which for each y_star_$bar_i \pm 1.96\sqrt{}$var(y_star)
            - 95% confidence interval = z-score 1.96

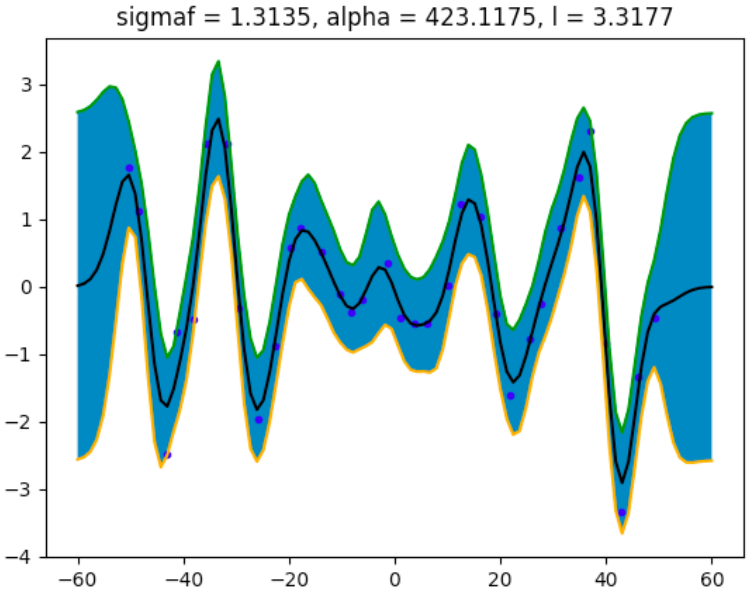


- I initialize the parameter to sigmaf = 1, alpha = 1, l = 1, finding that some points are not really fitted to the black line
- the green line is the upper bound of the 95% confidence interval
- the yellow line is the lower bound of the 95% confidence interval

hw5q1_2: Optimize the kernel parameters by minimizing negative marginal log-likelihood, and visualize the result again. (You can use scipy.optimize.minimize to optimize the parameters.)

```
# q1_2 #fun = lambda C, sigmaf, alpha, l: (0.5*np.log(C)) + (0.5*Y.T @ C.inv
def C(xa, xb, sigmaf, alpha, l):
    numerator = np.abs(xa - xb) * np.abs(xa - xb)
    denominator = 2 * alpha * l * l
    return sigmaf * sigmaf * np.power(1 + numerator / denominator, -alpha)

def fun(x, *args):
    # for arg in args:
    #     print("arg {}")
    #print("args {}".format(args))
    X, Y = args
    print("\nX {}\nY {}".format(X, Y))
    sigmaf, alpha, l = x
    print("sigmaf {} alpha {} l {}".format(sigmaf, alpha, l))
    K = np.zeros((len(X), len(X)))
    for i in range(len(X)):
        for j in range(len(X)):
            K[i][j] = C(X[i], X[j], sigmaf, alpha, l)
            if i == j:
                K[i][j] += 0.2
    return (0.5*np.log(np.linalg.det(K))) + (0.5*Y.T @ np.linalg.inv(K) @ Y)
res = minimize(fun, x0=[1, 1, 1], args=(X, Y))
print("res: {}".format(res.x))

# kernel
sample_count = 100
K = compute_K(X, Y, res.x[0], res.x[1], res.x[2])
X_lin = np.linspace(-60, 60, num=sample_count)
Y_lin = np.zeros(sample_count)
var_y_star = np.zeros(sample_count)
interval_upper = np.zeros(sample_count)
interval_lower = np.zeros(sample_count)
for i in range(len(X_lin)):
    K_star = compute_K_star(X, Y, X_lin[i], res.x[0], res.x[1], res.x[2])
    K_star_star = rational_quadratic_kernel(X_lin[i], X_lin[i], res.x[0], re
    y_star_bar = K_star@np.linalg.inv(K)@Y
    Y_lin[i] = y_star_bar
    var_y_star[i] = K_star_star - K_star@np.linalg.inv(K)@np.transpose(K_sta
for i in range(sample_count):
    interval_upper[i] = Y_lin[i] + 1.96*np.sqrt(var_y_star[i])
    interval_lower[i] = Y_lin[i] - 1.96*np.sqrt(var_y_star[i])

# visualization
plt.title("sigmaf = %0.4f, alpha = %0.4f, l = %0.4f" %(res.x[0], res.x[1], r
plt.plot(X, Y, 'b.')
plt.plot(X_lin, Y_lin, 'black')
plt.fill_between(X_lin, interval_upper, interval_lower)
plt.plot(X_lin, Y_lin + 1.96*np.sqrt(var_y_star), 'green')
plt.plot(X_lin, Y_lin - 1.96*np.sqrt(var_y_star), 'orange')
plt.show()
```

- I recompute the kernel K, K_star, K_star_star using the optimize parameters, and then visualize the result again



- optimize paramters: sigmaf = 1.3135, alpha = 423.1175, l = 3.3177
- I find that all the data points are more fitted to the black line(mean of the function), the upper bound(green line) and the lower bound(yellow line) are closer to the black line.
- The 95%confidence interval becomes tighter as we find the optimize parameters.
- The function I used:
  - fun(x, *args)
    - a function to be passed into scipy.optimize.minimize
    - x is an array containing our parameters: sigmaf, alpha, l
    - *args: the fixed parameter, here it's our data points (X,Y)
  - res = minimize(fun, x0=[1, 1, 1], args=(X, Y))
    - fun: a callable function
    - x0: the initial value of our parameters sigmaf, alpha, l
    - args: our data points (X,Y)
    - res: scipy.optimize.minimize returns our optimized parameters [sigmaf, alpha, l]

hw5q2

- Question： SVM on MNIST dataset
  - We use the library "libsvm" which is developed by Professor Chih-Jen Lin in NTU to perform classification on MNIST dataset
- training
  - dataset: 5000picture*784pixel
    - each pixel's value is between 0~1
  - label: 5000*1(only contain digit 1~5)
- test
  - dataset: 2500picture*784pixel
    - each pixel's value is between 0~1
  - label: 2500*1(only contain digit1~5)

hw5q2_1: Use different kernel functions (linear, polynomial, and RBF kernels) and have comparison between their performance.

- kernel function
  - an mathematical tool
  - simply and optimize the complicated computation caused by feature transformation
  - In simple word
    - dot product of 2 vectors after feature transformation

        - dot product can be viewed as “similarity”. If two vectors are in the same direction, the dot product is positive meaning that they’re similarity is high. Else if two vectors are not in the same direction, the dot product is negative meaning that they’re similarity is low.
        - In kernel method, Xn = Xm gives the maximum value indicating that the 2 vectors have the highest similarity.
- Polynomial kernel is equivalent to “multiplication of 2 new features after power of d non-linear transformation”
- RBF kernel is equivalent to “multiplication of 2 new features after power of infinity non-linear transformation”

- libsvm’s function “svm_train” performs training
    - first we pass training label y
        - dim: 5000*1
    - pass training data x
        - dim: 5000*784
    - pass parameter to the function
        - -t means which kernel we use
            - 0 = linear kernel
            - 1 = polynomial kernel
            - 2 = RBF kernel
            - 3 = sigmoid kernel
            - 4 = precomputed kernel
        - -b means whether to compute the accuracy or not
            - 0 = no need to compute
            - 1 = yes, compute the accuracy
    - the function return a model to be used in the next step(prediction)
- libsvm’s function “svm_predict” performs prediction
    - first we pass test label yt
        - dim: 2500*1
    - pass test data xt
        - dim: 2500*784
    - pass the model which has already been trained by training data and training label to perform prediction
    - function returns:
        - 2500 predictd test label
        - the accuracy of the prediction, MSE, correlation coefficient
        - p_vals: whether the prediction is reliable, it’s like the statistical term “p-value”

linear kernel:

- k(x, y) = $x^T y$ + c
- A basic kernel method
- perform dot product on two data and then add a constant term

```
model = svm_train(y, x, '-t 0 -b 1')
p_label, p_acc, p_val = svm_predict(yt, xt, model)
```

```
In [32]: # linear kernel
         #prob = svm_problem(y, x)
         #param = svm_parameter('-t 0 -b 1')
         model = svm_train(y, x, '-t 0 -b 1')
         p_label, p_acc, p_val = svm_predict(yt, xt, model)
         #print(p_label)

         Model supports probability estimates, but disabled in predicton.
         Accuracy = 95.08% (2377/2500) (classification)
```

- The accuracy is 95.08%, which is the third place of the 3 kernel methods.

polynomial kernel

- k(x, y) = $(\alpha x^T y + c)^d$

- (perform dot product on two data then be multiplied by a coefficient α then plus a constant c) power of d

```
model = svm_train(y, x, '-t 1 -b 1')
p_label, p_acc, p_val = svm_predict(yt, xt, model)
```

```
In [33]: # polynomial kernel
         model = svm_train(y, x, '-t 1 -b 1')
         p_label, p_acc, p_val = svm_predict(yt, xt, model)
         #print(p_label)

         Model supports probability estimates, but disabled in predicton.
         Accuracy = 34.68% (867/2500) (classification)
```

- At first, I'm wondering why the accuracy is so low(34.68%), and I think that maybe the polynomial kernel is not suitable for this dataset.
- Then, I look up the parameter specification and then find a parameter "-r" which accept a integer indicating the kernel method's coefficient α. I set $-r1$ which makes $\alpha = 1$ => problem solved
- The accuracy is 95.76% now.

```
In [8]: # polynomial kernel
        model = svm_train(y, x, '-t 1 -b 1 -r 1')
        p_label, p_acc, p_val = svm_predict(yt, xt, model)
        #print(p_label)

        Model supports probability estimates, but disabled in predicton.
        Accuracy = 95.76% (2394/2500) (classification)
```

RBF(radial basis function) kernel

- k(x, y) = $e^{\frac{-||x-y||^2}{2\sigma^2}} = e^{-\gamma||x-y||^2}$
- gamma is a coefficient which I set it to 1/784

```
model = svm_train(y, x, '-t 2 -b 1')
p_label, p_acc, p_val = svm_predict(yt, xt, model)
```

```
In [34]: # RBF kernel
         model = svm_train(y, x, '-t 2 -b 1')
         p_label, p_acc, p_val = svm_predict(yt, xt, model)
         #print(p_label)

         Model supports probability estimates, but disabled in predicton.
         Accuracy = 95.32% (2383/2500) (classification)
```

- The accuracy is 95.32%, which is the second best among the 3 kernel method

hw5q2_2: Please use C-SVC (you can choose by setting parameters in the function input, C-SVC is soft-margin SVM). Since there are some parameters you need to tune for, please do the grid search for finding parameters of best performing model. For instance, in C-SVC you have a parameter C, and if you use RBF kernel you have another parameter γ, you can search for a set of (C, γ) which gives you best performance in cross-validation. (There are lots of sources on internet, just google for it.)

linear kernel

```
# q2_2 linear kernel
C = [2**-5, 2**-3, 2**-1, 2**1, 2**3, 2**5, 2**7, 2**9, 2**11, 2**13, 2**15]
max_acc = 0.0
max_c = 0.0
for c in C:
    model = svm_train(y, x, '-t 0 -b 1 -s 0 -c {} -v 5'.format(c))
    print('c = {}, accuracy = {}'.format(c, model))
    if model > max_acc:
        max_acc = model
        max_c = c
print("max_acc", max_acc, "max_c", max_c)
model = svm_train(y, x, '-t 0 -b 1 -s 0 -c {}'.format(max_c))
p_label, p_acc, p_val = svm_predict(yt, xt, model)
```

- SVM requires the solution of the following optimization problem
  - $min_{w,b,\epsilon} \frac{1}{2} w^T w + C\Sigma_{i=1}^{l} \epsilon_i$
  - l: number of data, here 5000(training data)
- C = [2**-5, 2**-3, ..., 2**15] is the coefficient before $\Sigma_{i=1}^{l} \epsilon_i$
  - These values is recommended by the author mentioned in “guide.pdf”
- Step
  - set max_acc = 0.0, max_c = 0.0
  - we want to perform “grid search”, so we try c in C once at a time (for c in C)
  - For “svm_train”
    - pass training label
    - pass training data
    - pass parameter
      - -t 0 means linear kernel
      - -b 1 means to compute accuracy
      - -s 0 means using C-SVC
      - -c {} means try c in C one by one
      - -v 5 means 5-fold cross validation
- we update max_c when currently used c gives a higher accuracy
- After cross validation, we train the model using the c (max_c) which gives the highest accuracy
- Finally, we perform prediction based on the model we’ve trained
-

```
In [4]: # q2_2 linear kernel
C = [2**-5, 2**-3, 2**-1, 2**1, 2**3, 2**5, 2**7, 2**9, 2**11, 2**13, 2**15]
#C = [2, 8]
max_acc = 0.0
max_c = 0.0
for c in C:
    model = svm_train(y, x, '-t 0 -b 1 -s 0 -c {} -v 5'.format(c))
    print('c = {}, accuracy = {}'.format(c, model))
    if model > max_acc:
        max_acc = model
        max_c = c
print("max_acc", max_acc, "max_c", max_c)
model = svm_train(y, x, '-t 0 -b 1 -s 0 -c {}'.format(max_c))
p_label, p_acc, p_val = svm_predict(yt, xt, model)

Cross Validation Accuracy = 97.16%
c = 0.03125, accuracy = 97.16
Cross Validation Accuracy = 97.08%
c = 0.125, accuracy = 97.08
Cross Validation Accuracy = 96.42%
c = 0.5, accuracy = 96.41999999999999
Cross Validation Accuracy = 96.64%
c = 2, accuracy = 96.64
Cross Validation Accuracy = 96.44%
c = 8, accuracy = 96.44
Cross Validation Accuracy = 96.84%
c = 32, accuracy = 96.84
Cross Validation Accuracy = 96.38%
c = 128, accuracy = 96.38
Cross Validation Accuracy = 96.46%
c = 512, accuracy = 96.46000000000001
Cross Validation Accuracy = 96.52%
```

```
    if model > max_acc:
        max_acc = model
        max_c = c
print("max_acc", max_acc, "max_c", max_c)
model = svm_train(y, x, '-t 0 -b 1 -s 0 -c {}'.format(max_c))
p_label, p_acc, p_val = svm_predict(yt, xt, model)
```

```
Cross Validation Accuracy = 97.16%
c = 0.03125, accuracy = 97.16
Cross Validation Accuracy = 97.08%
c = 0.125, accuracy = 97.08
Cross Validation Accuracy = 96.42%
c = 0.5, accuracy = 96.41999999999999
Cross Validation Accuracy = 96.64%
c = 2, accuracy = 96.64
Cross Validation Accuracy = 96.44%
c = 8, accuracy = 96.44
Cross Validation Accuracy = 96.84%
c = 32, accuracy = 96.84
Cross Validation Accuracy = 96.38%
c = 128, accuracy = 96.38
Cross Validation Accuracy = 96.46%
c = 512, accuracy = 96.46000000000001
Cross Validation Accuracy = 96.52%
c = 2048, accuracy = 96.52
Cross Validation Accuracy = 96.34%
c = 8192, accuracy = 96.34
Cross Validation Accuracy = 96.66%
c = 32768, accuracy = 96.66
max_acc 97.16 max_c 0.03125
Model supports probability estimates, but disabled in predicton.
Accuracy = 96% (2400/2500) (classification)
```

- we got 96% accuracy.

RBF

```
# q2_2 RBF kernel
C = [2**-5, 2**-3, 2**-1, 2**1, 2**3, 2**5, 2**7, 2**9, 2**11, 2**13, 2**15]
G = [2**-15, 2**-13, 2**-11, 2**-9, 2**-7, 2**-5, 2**-3, 2**-1, 2**1, 2**3]
max_acc = 0.0
max_c = 0.0
max_gamma = 0.0
for c in C:
    for gamma in G:
        model = svm_train(y, x, '-t 2 -b 1 -s 0 -c {} -g {} -v 5'.format(c, gamm
        print('c = {}, gamma = {}, accuracy = {}'.format(c, gamma, model))
        if model > max_acc:
            max_acc = model
            max_c = c
            max_gamma = gamma
print("max_acc", max_acc, "max_c", max_c, "max_gamma", max_gamma)
model = svm_train(y, x, '-t 2 -b 1 -s 0 -c {} -g {}'.format(max_c, max_gamma))
p_label, p_acc, p_val = svm_predict(yt, xt, model)
```

- C is the coefficient of $\Sigma\epsilon_i$
  - C = [2**-5, 2**-3, ..., 2**15]
  - These values are recommended by the author mentioned in “guide.pdf”
- G is the term in RBF kernel, k(x,y)=$e^{-\gamma||x-y||^2}$
  - G = [2**-15, 2**-13, ..., 2**3]
  - These values are recommended by the author mentioned in “guide.pdf”
- Step
  - set max_acc = 0.0, max_c = 0.0, max_gamma = 0.0
  - Just like mentioned above for the linear method, we use grid search(11*10= 110 pairs of (c, gamma)) and cross validation to find the parameter c, gamma which give the highest accuracy
  - Then we use the max_c, max_gamma to train again our model which will be applied in the prediction step
  - Finally, we use the trained model to predict our test label.

In [6]:
```
# q2_2 RBF kernel
C = [2**-5, 2**-3, 2**-1, 2**1, 2**3, 2**5, 2**7, 2**9, 2**11, 2**13, 2**15]
G = [2**-15, 2**-13, 2**-11, 2**-9, 2**-7, 2**-5, 2**-3, 2**-1, 2**1, 2**3]
# C = [2, 8]
# G = [0.5]
max_acc = 0.0
max_c = 0.0
max_gamma = 0.0
for c in C:
    for gamma in G:
        model = svm_train(y, x, '-t 2 -b 1 -s 0 -c {} -g {} -v 5'.format(c, gamma))
        print('c = {}, gamma = {}, accuracy = {}'.format(c, gamma, model))
        if model > max_acc:
            max_acc = model
            max_c = c
            max_gamma = gamma
print("max_acc", max_acc, "max_c", max_c, "max_gamma", max_gamma)
model = svm_train(y, x, '-t 2 -b 1 -s 0 -c {} -g {}'.format(max_c, max_gamma))
p_label, p_acc, p_val = svm_predict(yt, xt, model)
```

```
Cross Validation Accuracy = 12.38%
c = 0.03125, gamma = 3.0517578125e-05, accuracy = 12.379999999999999
Cross Validation Accuracy = 38.44%
c = 0.03125, gamma = 0.0001220703125, accuracy = 38.440000000000005
Cross Validation Accuracy = 83.62%
c = 0.03125, gamma = 0.00048828125, accuracy = 83.62
Cross Validation Accuracy = 93.34%
c = 0.03125, gamma = 0.001953125, accuracy = 93.34
Cross Validation Accuracy = 95.1%
c = 0.03125, gamma = 0.0078125, accuracy = 95.1
Cross Validation Accuracy = 95.5%
c = 0.03125, gamma = 0.03125, accuracy = 95.5
```

```
max_gamma = 0.0
for c in C:
    for gamma in G:
        model = svm_train(y, x, '-t 2 -b 1 -s 0 -c {} -g {} -v 5'.format(c, gamma))
        print('c = {}, gamma = {}, accuracy = {}'.format(c, gamma, model))
        if model > max_acc:
            max_acc = model
            max_c = c
            max_gamma = gamma
print("max_acc", max_acc, "max_c", max_c, "max_gamma", max_gamma)
model = svm_train(y, x, '-t 2 -b 1 -s 0 -c {} -g {}'.format(max_c, max_gamma))
p_label, p_acc, p_val = svm_predict(yt, xt, model)
```

```
Cross Validation Accuracy = 95.5%
c = 0.03125, gamma = 0.03125, accuracy = 95.5
Cross Validation Accuracy = 35%
c = 0.03125, gamma = 0.125, accuracy = 35.0
Cross Validation Accuracy = 19.2%
c = 0.03125, gamma = 0.5, accuracy = 19.2
Cross Validation Accuracy = 20%
c = 0.03125, gamma = 2, accuracy = 20.0
Cross Validation Accuracy = 20%
c = 0.03125, gamma = 8, accuracy = 20.0
Cross Validation Accuracy = 39.24%
c = 0.125, gamma = 3.0517578125e-05, accuracy = 39.24
Cross Validation Accuracy = 83.46%
c = 0.125, gamma = 0.0001220703125, accuracy = 83.46000000000001
Cross Validation Accuracy = 93.3%
c = 0.125, gamma = 0.00048828125, accuracy = 93.30000000000001
Cross Validation Accuracy = 95.4%
c = 0.125, gamma = 0.001953125, accuracy = 95.39999999999999
Cross Validation Accuracy = 96.56%
```

Save and Checkpoint

```
max_gamma = 0.0
for c in C:
    for gamma in G:
        model = svm_train(y, x, '-t 2 -b 1 -s 0 -c {} -g {} -v 5'.format(c, gamma))
        print('c = {}, gamma = {}, accuracy = {}'.format(c, gamma, model))
        if model > max_acc:
            max_acc = model
            max_c = c
            max_gamma = gamma
print("max_acc", max_acc, "max_c", max_c, "max_gamma", max_gamma)
model = svm_train(y, x, '-t 2 -b 1 -s 0 -c {} -g {}'.format(max_c, max_gamma))
p_label, p_acc, p_val = svm_predict(yt, xt, model)
```

```
Cross Validation Accuracy = 95.4%
c = 0.125, gamma = 0.001953125, accuracy = 95.39999999999999
Cross Validation Accuracy = 96.56%
c = 0.125, gamma = 0.0078125, accuracy = 96.56
Cross Validation Accuracy = 97.28%
c = 0.125, gamma = 0.03125, accuracy = 97.28
Cross Validation Accuracy = 42.32%
c = 0.125, gamma = 0.125, accuracy = 42.32
Cross Validation Accuracy = 20.3%
c = 0.125, gamma = 0.5, accuracy = 20.3
Cross Validation Accuracy = 20%
c = 0.125, gamma = 2, accuracy = 20.0
Cross Validation Accuracy = 20%
c = 0.125, gamma = 8, accuracy = 20.0
Cross Validation Accuracy = 82.76%
c = 0.5, gamma = 3.0517578125e-05, accuracy = 82.76
Cross Validation Accuracy = 93.32%
c = 0.5, gamma = 0.0001220703125, accuracy = 93.32000000000001
Cross Validation Accuracy = 95.14%
```

```
max_gamma = 0.0
for c in C:
    for gamma in G:
        model = svm_train(y, x, '-t 2 -b 1 -s 0 -c {} -g {} -v 5'.format(c, gamma))
        print('c = {}, gamma = {}, accuracy = {}'.format(c, gamma, model))
        if model > max_acc:
            max_acc = model
            max_c = c
            max_gamma = gamma
print("max_acc", max_acc, "max_c", max_c, "max_gamma", max_gamma)
model = svm_train(y, x, '-t 2 -b 1 -s 0 -c {} -g {}'.format(max_c, max_gamma))
p_label, p_acc, p_val = svm_predict(yt, xt, model)
```

```
Cross Validation Accuracy = 93.32%
c = 0.5, gamma = 0.0001220703125, accuracy = 93.32000000000001
Cross Validation Accuracy = 95.14%
c = 0.5, gamma = 0.00048828125, accuracy = 95.14
Cross Validation Accuracy = 96.46%
c = 0.5, gamma = 0.001953125, accuracy = 96.46000000000001
Cross Validation Accuracy = 97.4%
c = 0.5, gamma = 0.0078125, accuracy = 97.39999999999999
Cross Validation Accuracy = 98.26%
c = 0.5, gamma = 0.03125, accuracy = 98.26
Cross Validation Accuracy = 79.04%
c = 0.5, gamma = 0.125, accuracy = 79.03999999999999
Cross Validation Accuracy = 31.92%
c = 0.5, gamma = 0.5, accuracy = 31.919999999999998
Cross Validation Accuracy = 19.76%
c = 0.5, gamma = 2, accuracy = 19.759999999999998
Cross Validation Accuracy = 20%
c = 0.5, gamma = 8, accuracy = 20.0
Cross Validation Accuracy = 93.42%
```

```
max_gamma = 0.0
for c in C:
    for gamma in G:
        model = svm_train(y, x, '-t 2 -b 1 -s 0 -c {} -g {} -v 5'.format(c, gamma))
        print('c = {}, gamma = {}, accuracy = {}'.format(c, gamma, model))
        if model > max_acc:
            max_acc = model
            max_c = c
            max_gamma = gamma
print("max_acc", max_acc, "max_c", max_c, "max_gamma", max_gamma)
model = svm_train(y, x, '-t 2 -b 1 -s 0 -c {} -g {}'.format(max_c, max_gamma))
p_label, p_acc, p_val = svm_predict(yt, xt, model)
```

```
Cross Validation Accuracy = 20%
c = 0.5, gamma = 8, accuracy = 20.0
Cross Validation Accuracy = 93.42%
c = 2, gamma = 3.0517578125e-05, accuracy = 93.42
Cross Validation Accuracy = 95.34%
c = 2, gamma = 0.0001220703125, accuracy = 95.34
Cross Validation Accuracy = 96.44%
c = 2, gamma = 0.00048828125, accuracy = 96.44
Cross Validation Accuracy = 97.12%
c = 2, gamma = 0.001953125, accuracy = 97.11999999999999
Cross Validation Accuracy = 98.12%
c = 2, gamma = 0.0078125, accuracy = 98.11999999999999
Cross Validation Accuracy = 98.68%
c = 2, gamma = 0.03125, accuracy = 98.68
Cross Validation Accuracy = 96.32%
c = 2, gamma = 0.125, accuracy = 96.32
Cross Validation Accuracy = 33.64%
c = 2, gamma = 0.5, accuracy = 33.64
Cross Validation Accuracy = 24.66%
c = 2, gamma = 2, accuracy = 24.66
```

```
max_gamma = 0.0
for c in C:
    for gamma in G:
        model = svm_train(y, x, '-t 2 -b 1 -s 0 -c {} -g {} -v 5'.format(c, gamma))
        print('c = {}, gamma = {}, accuracy = {}'.format(c, gamma, model))
        if model > max_acc:
            max_acc = model
            max_c = c
            max_gamma = gamma
print("max_acc", max_acc, "max_c", max_c, "max_gamma", max_gamma)
model = svm_train(y, x, '-t 2 -b 1 -s 0 -c {} -g {}'.format(max_c, max_gamma))
p_label, p_acc, p_val = svm_predict(yt, xt, model)
```

```
Cross Validation Accuracy = 33.64%
c = 2, gamma = 0.5, accuracy = 33.64
Cross Validation Accuracy = 24.66%
c = 2, gamma = 2, accuracy = 24.66
Cross Validation Accuracy = 20%
c = 2, gamma = 8, accuracy = 20.0
Cross Validation Accuracy = 95.26%
c = 8, gamma = 3.0517578125e-05, accuracy = 95.26
Cross Validation Accuracy = 96.3%
c = 8, gamma = 0.0001220703125, accuracy = 96.3
Cross Validation Accuracy = 97%
c = 8, gamma = 0.00048828125, accuracy = 97.0
Cross Validation Accuracy = 97.48%
c = 8, gamma = 0.001953125, accuracy = 97.48
Cross Validation Accuracy = 98.12%
c = 8, gamma = 0.0078125, accuracy = 98.11999999999999
Cross Validation Accuracy = 98.62%
c = 8, gamma = 0.03125, accuracy = 98.61999999999999
Cross Validation Accuracy = 96.2%
c = 8, gamma = 0.125, accuracy = 96.2
```

```
max_gamma = 0.0
for c in C:
    for gamma in G:
        model = svm_train(y, x, '-t 2 -b 1 -s 0 -c {} -g {} -v 5'.format(c, gamma))
        print('c = {}, gamma = {}, accuracy = {}'.format(c, gamma, model))
        if model > max_acc:
            max_acc = model
            max_c = c
            max_gamma = gamma
print("max_acc", max_acc, "max_c", max_c, "max_gamma", max_gamma)
model = svm_train(y, x, '-t 2 -b 1 -s 0 -c {} -g {}'.format(max_c, max_gamma))
p_label, p_acc, p_val = svm_predict(yt, xt, model)
```

```
Cross Validation Accuracy = 98.62%
c = 8, gamma = 0.03125, accuracy = 98.61999999999999
Cross Validation Accuracy = 96.2%
c = 8, gamma = 0.125, accuracy = 96.2
Cross Validation Accuracy = 37.36%
c = 8, gamma = 0.5, accuracy = 37.36
Cross Validation Accuracy = 24.62%
c = 8, gamma = 2, accuracy = 24.62
Cross Validation Accuracy = 20%
c = 8, gamma = 8, accuracy = 20.0
Cross Validation Accuracy = 96.26%
c = 32, gamma = 3.0517578125e-05, accuracy = 96.26
Cross Validation Accuracy = 96.82%
c = 32, gamma = 0.0001220703125, accuracy = 96.82
Cross Validation Accuracy = 97.2%
c = 32, gamma = 0.00048828125, accuracy = 97.2
Cross Validation Accuracy = 97.5%
c = 32, gamma = 0.001953125, accuracy = 97.5
Cross Validation Accuracy = 98.16%
c = 32, gamma = 0.0078125, accuracy = 98.16
```

```
max_gamma = 0.0
for c in C:
    for gamma in G:
        model = svm_train(y, x, '-t 2 -b 1 -s 0 -c {} -g {} -v 5'.format(c, gamma))
        print('c = {}, gamma = {}, accuracy = {}'.format(c, gamma, model))
        if model > max_acc:
            max_acc = model
            max_c = c
            max_gamma = gamma
print("max_acc", max_acc, "max_c", max_c, "max_gamma", max_gamma)
model = svm_train(y, x, '-t 2 -b 1 -s 0 -c {} -g {}'.format(max_c, max_gamma))
p_label, p_acc, p_val = svm_predict(yt, xt, model)
```

```
Cross Validation Accuracy = 97.5%
c = 32, gamma = 0.001953125, accuracy = 97.5
Cross Validation Accuracy = 98.16%
c = 32, gamma = 0.0078125, accuracy = 98.16
Cross Validation Accuracy = 98.56%
c = 32, gamma = 0.03125, accuracy = 98.56
Cross Validation Accuracy = 96.14%
c = 32, gamma = 0.125, accuracy = 96.14
Cross Validation Accuracy = 32.12%
c = 32, gamma = 0.5, accuracy = 32.12
Cross Validation Accuracy = 24.38%
c = 32, gamma = 2, accuracy = 24.38
Cross Validation Accuracy = 20%
c = 32, gamma = 8, accuracy = 20.0
Cross Validation Accuracy = 96.84%
c = 128, gamma = 3.0517578125e-05, accuracy = 96.84
Cross Validation Accuracy = 97.02%
c = 128, gamma = 0.0001220703125, accuracy = 97.02
Cross Validation Accuracy = 97.38%
```

```
max_gamma = 0.0
for c in C:
    for gamma in G:
        model = svm_train(y, x, '-t 2 -b 1 -s 0 -c {} -g {} -v 5'.format(c, gamma))
        print('c = {}, gamma = {}, accuracy = {}'.format(c, gamma, model))
        if model > max_acc:
            max_acc = model
            max_c = c
            max_gamma = gamma
print("max_acc", max_acc, "max_c", max_c, "max_gamma", max_gamma)
model = svm_train(y, x, '-t 2 -b 1 -s 0 -c {} -g {}'.format(max_c, max_gamma))
p_label, p_acc, p_val = svm_predict(yt, xt, model)
```

```
Cross Validation Accuracy = 97.02%
c = 128, gamma = 0.0001220703125, accuracy = 97.02
Cross Validation Accuracy = 97.38%
c = 128, gamma = 0.00048828125, accuracy = 97.38
Cross Validation Accuracy = 97.58%
c = 128, gamma = 0.001953125, accuracy = 97.58
Cross Validation Accuracy = 98.14%
c = 128, gamma = 0.0078125, accuracy = 98.14
Cross Validation Accuracy = 98.66%
c = 128, gamma = 0.03125, accuracy = 98.66
Cross Validation Accuracy = 96.28%
c = 128, gamma = 0.125, accuracy = 96.28
Cross Validation Accuracy = 35.24%
c = 128, gamma = 0.5, accuracy = 35.24
Cross Validation Accuracy = 24.5%
c = 128, gamma = 2, accuracy = 24.5
Cross Validation Accuracy = 20%
c = 128, gamma = 8, accuracy = 20.0
Cross Validation Accuracy = 97.32%
c = 512, gamma = 3.0517578125e-05, accuracy = 97.32
```

```
max_gamma = 0.0
for c in C:
    for gamma in G:
        model = svm_train(y, x, '-t 2 -b 1 -s 0 -c {} -g {} -v 5'.format(c, gamma))
        print('c = {}, gamma = {}, accuracy = {}'.format(c, gamma, model))
        if model > max_acc:
            max_acc = model
            max_c = c
            max_gamma = gamma
print("max_acc", max_acc, "max_c", max_c, "max_gamma", max_gamma)
model = svm_train(y, x, '-t 2 -b 1 -s 0 -c {} -g {}'.format(max_c, max_gamma))
p_label, p_acc, p_val = svm_predict(yt, xt, model)
```

```
Cross Validation Accuracy = 20%
c = 128, gamma = 8, accuracy = 20.0
Cross Validation Accuracy = 97.32%
c = 512, gamma = 3.0517578125e-05, accuracy = 97.32
Cross Validation Accuracy = 97.1%
c = 512, gamma = 0.0001220703125, accuracy = 97.1
Cross Validation Accuracy = 96.94%
c = 512, gamma = 0.00048828125, accuracy = 96.94
Cross Validation Accuracy = 97.46%
c = 512, gamma = 0.001953125, accuracy = 97.46000000000001
Cross Validation Accuracy = 98.28%
c = 512, gamma = 0.0078125, accuracy = 98.28
Cross Validation Accuracy = 98.58%
c = 512, gamma = 0.03125, accuracy = 98.58
Cross Validation Accuracy = 96.12%
c = 512, gamma = 0.125, accuracy = 96.12
Cross Validation Accuracy = 33.42%
c = 512, gamma = 0.5, accuracy = 33.42
Cross Validation Accuracy = 24.78%
c = 512, gamma = 2, accuracy = 24.779999999999998
```

```
max_gamma = 0.0
for c in C:
    for gamma in G:
        model = svm_train(y, x, '-t 2 -b 1 -s 0 -c {} -g {} -v 5'.format(c, gamma))
        print('c = {}, gamma = {}, accuracy = {}'.format(c, gamma, model))
        if model > max_acc:
            max_acc = model
            max_c = c
            max_gamma = gamma
print("max_acc", max_acc, "max_c", max_c, "max_gamma", max_gamma)
model = svm_train(y, x, '-t 2 -b 1 -s 0 -c {} -g {}'.format(max_c, max_gamma))
p_label, p_acc, p_val = svm_predict(yt, xt, model)
```

```
Cross Validation Accuracy = 33.42%
c = 512, gamma = 0.5, accuracy = 33.42
Cross Validation Accuracy = 24.78%
c = 512, gamma = 2, accuracy = 24.779999999999998
Cross Validation Accuracy = 20%
c = 512, gamma = 8, accuracy = 20.0
Cross Validation Accuracy = 97.02%
c = 2048, gamma = 3.0517578125e-05, accuracy = 97.02
Cross Validation Accuracy = 96.66%
c = 2048, gamma = 0.0001220703125, accuracy = 96.66
Cross Validation Accuracy = 97.2%
c = 2048, gamma = 0.00048828125, accuracy = 97.2
Cross Validation Accuracy = 97.6%
c = 2048, gamma = 0.001953125, accuracy = 97.6
Cross Validation Accuracy = 98.16%
c = 2048, gamma = 0.0078125, accuracy = 98.16
Cross Validation Accuracy = 98.72%
c = 2048, gamma = 0.03125, accuracy = 98.72
Cross Validation Accuracy = 95.96%
c = 2048, gamma = 0.125, accuracy = 95.96000000000001
```

```
max_gamma = 0.0
for c in C:
    for gamma in G:
        model = svm_train(y, x, '-t 2 -b 1 -s 0 -c {} -g {} -v 5'.format(c, gamma))
        print('c = {}, gamma = {}, accuracy = {}'.format(c, gamma, model))
        if model > max_acc:
            max_acc = model
            max_c = c
            max_gamma = gamma
print("max_acc", max_acc, "max_c", max_c, "max_gamma", max_gamma)
model = svm_train(y, x, '-t 2 -b 1 -s 0 -c {} -g {}'.format(max_c, max_gamma))
p_label, p_acc, p_val = svm_predict(yt, xt, model)
```

```
Cross Validation Accuracy = 98.72%
c = 2048, gamma = 0.03125, accuracy = 98.72
Cross Validation Accuracy = 95.96%
c = 2048, gamma = 0.125, accuracy = 95.96000000000001
Cross Validation Accuracy = 33.64%
c = 2048, gamma = 0.5, accuracy = 33.64
Cross Validation Accuracy = 24.72%
c = 2048, gamma = 2, accuracy = 24.72
Cross Validation Accuracy = 20%
c = 2048, gamma = 8, accuracy = 20.0
Cross Validation Accuracy = 96.72%
c = 8192, gamma = 3.0517578125e-05, accuracy = 96.72
Cross Validation Accuracy = 96.82%
c = 8192, gamma = 0.0001220703125, accuracy = 96.82
Cross Validation Accuracy = 97.16%
c = 8192, gamma = 0.00048828125, accuracy = 97.16
Cross Validation Accuracy = 97.84%
c = 8192, gamma = 0.001953125, accuracy = 97.84
Cross Validation Accuracy = 98.16%
c = 8192, gamma = 0.0078125, accuracy = 98.16
```

Save and Checkpoint

```
max_gamma = 0.0
for c in C:
    for gamma in G:
        model = svm_train(y, x, '-t 2 -b 1 -s 0 -c {} -g {} -v 5'.format(c, gamma))
        print('c = {}, gamma = {}, accuracy = {}'.format(c, gamma, model))
        if model > max_acc:
            max_acc = model
            max_c = c
            max_gamma = gamma
print("max_acc", max_acc, "max_c", max_c, "max_gamma", max_gamma)
model = svm_train(y, x, '-t 2 -b 1 -s 0 -c {} -g {}'.format(max_c, max_gamma))
p_label, p_acc, p_val = svm_predict(yt, xt, model)
```

```
Cross Validation Accuracy = 97.84%
c = 8192, gamma = 0.001953125, accuracy = 97.84
Cross Validation Accuracy = 98.16%
c = 8192, gamma = 0.0078125, accuracy = 98.16
Cross Validation Accuracy = 98.58%
c = 8192, gamma = 0.03125, accuracy = 98.58
Cross Validation Accuracy = 96.14%
c = 8192, gamma = 0.125, accuracy = 96.14
Cross Validation Accuracy = 33.42%
c = 8192, gamma = 0.5, accuracy = 33.42
Cross Validation Accuracy = 24.62%
c = 8192, gamma = 2, accuracy = 24.62
Cross Validation Accuracy = 20%
c = 8192, gamma = 8, accuracy = 20.0
Cross Validation Accuracy = 96.66%
c = 32768, gamma = 3.0517578125e-05, accuracy = 96.66
Cross Validation Accuracy = 96.64%
c = 32768, gamma = 0.0001220703125, accuracy = 96.64

Cross Validation Accuracy = 97.22%
```

```
max_gamma = 0.0
for c in C:
    for gamma in G:
        model = svm_train(y, x, '-t 2 -b 1 -s 0 -c {} -g {} -v 5'.format(c, gamma))
        print('c = {}, gamma = {}, accuracy = {}'.format(c, gamma, model))
        if model > max_acc:
            max_acc = model
            max_c = c
            max_gamma = gamma
print("max_acc", max_acc, "max_c", max_c, "max_gamma", max_gamma)
model = svm_train(y, x, '-t 2 -b 1 -s 0 -c {} -g {}'.format(max_c, max_gamma))
p_label, p_acc, p_val = svm_predict(yt, xt, model)
```

```
Cross Validation Accuracy = 96.64%
c = 32768, gamma = 0.0001220703125, accuracy = 96.64

Cross Validation Accuracy = 97.22%
c = 32768, gamma = 0.00048828125, accuracy = 97.22
Cross Validation Accuracy = 97.66%
c = 32768, gamma = 0.001953125, accuracy = 97.66
Cross Validation Accuracy = 98.2%
c = 32768, gamma = 0.0078125, accuracy = 98.2
Cross Validation Accuracy = 98.48%
c = 32768, gamma = 0.03125, accuracy = 98.48
Cross Validation Accuracy = 96.16%
c = 32768, gamma = 0.125, accuracy = 96.16
Cross Validation Accuracy = 35.04%
c = 32768, gamma = 0.5, accuracy = 35.04
Cross Validation Accuracy = 24.7%
c = 32768, gamma = 2, accuracy = 24.7
Cross Validation Accuracy = 20%
c = 32768, gamma = 8, accuracy = 20.0
max_acc 98.72 max_c 2048 max_gamma 0.03125
```

```
max_gamma = 0.0
for c in C:
    for gamma in G:
        model = svm_train(y, x, '-t 2 -b 1 -s 0 -c {} -g {} -v 5'.format(c, gamma))
        print('c = {}, gamma = {}, accuracy = {}'.format(c, gamma, model))
        if model > max_acc:
            max_acc = model
            max_c = c
            max_gamma = gamma
print("max_acc", max_acc, "max_c", max_c, "max_gamma", max_gamma)
model = svm_train(y, x, '-t 2 -b 1 -s 0 -c {} -g {}'.format(max_c, max_gamma))
p_label, p_acc, p_val = svm_predict(yt, xt, model)
```

```
Cross Validation Accuracy = 97.22%
c = 32768, gamma = 0.00048828125, accuracy = 97.22
Cross Validation Accuracy = 97.66%
c = 32768, gamma = 0.001953125, accuracy = 97.66
Cross Validation Accuracy = 98.2%
c = 32768, gamma = 0.0078125, accuracy = 98.2
Cross Validation Accuracy = 98.48%
c = 32768, gamma = 0.03125, accuracy = 98.48
Cross Validation Accuracy = 96.16%
c = 32768, gamma = 0.125, accuracy = 96.16
Cross Validation Accuracy = 35.04%
c = 32768, gamma = 0.5, accuracy = 35.04
Cross Validation Accuracy = 24.7%
c = 32768, gamma = 2, accuracy = 24.7
Cross Validation Accuracy = 20%
c = 32768, gamma = 8, accuracy = 20.0
max_acc 98.72 max_c 2048 max_gamma 0.03125
Model supports probability estimates, but disabled in predicton.
Accuracy = 98.52% (2463/2500) (classification)
```

```
max_gamma = 0.0
for c in C:
    for gamma in G:
        model = svm_train(y, x, '-t 2 -b 1 -s 0 -c {} -g {} -v 5'.format(c, gamma))
        print('c = {}, gamma = {}, accuracy = {}'.format(c, gamma, model))
        if model > max_acc:
            max_acc = model
            max_c = c
            max_gamma = gamma
print("max_acc", max_acc, "max_c", max_c, "max_gamma", max_gamma)
model = svm_train(y, x, '-t 2 -b 1 -s 0 -c {} -g {}'.format(max_c, max_gamma))
p_label, p_acc, p_val = svm_predict(yt, xt, model)
```

```
Cross Validation Accuracy = 97.22%
c = 32768, gamma = 0.00048828125, accuracy = 97.22
Cross Validation Accuracy = 97.66%
c = 32768, gamma = 0.001953125, accuracy = 97.66
Cross Validation Accuracy = 98.2%
c = 32768, gamma = 0.0078125, accuracy = 98.2
Cross Validation Accuracy = 98.48%
c = 32768, gamma = 0.03125, accuracy = 98.48
Cross Validation Accuracy = 96.16%
c = 32768, gamma = 0.125, accuracy = 96.16
Cross Validation Accuracy = 35.04%
c = 32768, gamma = 0.5, accuracy = 35.04
Cross Validation Accuracy = 24.7%
c = 32768, gamma = 2, accuracy = 24.7
Cross Validation Accuracy = 20%
c = 32768, gamma = 8, accuracy = 20.0
max_acc 98.72 max_c 2048 max_gamma 0.03125
Model supports probability estimates, but disabled in predicton.
Accuracy = 98.52% (2463/2500) (classification)
```

- The pictures above show the 110 combinations of (c, gamma)'s results.
- we got 98.52% accuracy.
  - it's better than linear kernel's 96% accuracy

- This program runs for 10.5 hours, it’s really really long. I’ll try to find the way to speed it up afterwards.

hw5q2_3: Use linear kernel+RBF kernel together (therefore a new kernel function) and compare its performance with respect to others. You would need to find out how to use a user-defined kernel in libsvm.

```
import numpy as np
from libsvm.svmutil import *
from libsvm.svm import *

Y_train = open('Y_train.csv', 'r')
X_train = open('X_train.csv', 'r')
Y_test = open('Y_test.csv', 'r')
X_test = open('X_test.csv', 'r')

y = []
for line in Y_train.readlines():
    line = int(line)
    y.append(line)
#print(y)

x = []
for line in X_train.readlines():
    tmp = line.split(',')
    elem = []
    for pixel in tmp:
        pixel = float(pixel)
        elem.append(pixel)
    x.append(elem)
#print(x[0])

yt = []
for line in Y_test.readlines():
    line = int(line)
    yt.append(line)

xt = []
for line in X_test.readlines():
    tmp = line.split(',')
    elem = []
    for pixel in tmp:
        pixel = float(pixel)
        elem.append(pixel)
    xt.append(elem)

## # q2_3
def linear_kernel(xa, xb):
    xa = np.array(xa)
    xb = np.array(xb)
    #print("xa {}".format(xa))
    #print(xa.T @ xb)
    return xa.T @ xb

def RBF_kernel(xa, xb):
    gamma = 1/784
    xa_subtract_xb = [xa[i] - xb[i] for i in range(len(xa))]
    xa_subtract_xb = np.array(xa_subtract_xb)
    #xa_subtract_xb = np.array([a-b for (a, b) in zip(xa, xb)])
    #print("\nxa {}\nxb {}\n xa_subtract_xb {}".format(xa, xb, xa_subtract_xb))
    #print(xa_subtract_xb.T @ xa_subtract_xb/2/sigma/sigma)
    #print(xa_subtract_xb.T @ xa_subtract_xb)
    return np.exp(-1*gamma*(xa_subtract_xb.T @ xa_subtract_xb))

#print(linear_kernel([7,8,9], [1,2,3]))
#print(RBF_kernel([7,8,9], [1,2,3]))

beta = 0.2
K = []
train_count = len(x)
for i in range(train_count):
    tmp = []
    tmp.append(i+1)
    for j in range(train_count):
        k_linear = linear_kernel(x[i], x[j])
        k_RBF = RBF_kernel(x[i], x[j])
        tmp.append(k_linear + k_RBF)
    tmp[i+1] += beta
    K.append(tmp)
#print(K)
print("K has already been generated.")

K_test = []
test_count = len(xt)
```

```
for i in range(test_count):
    tmp = []
    tmp.append(i+1)
    for j in range(train_count):
        k_linear = linear_kernel(xt[i], x[j])
        k_RBF = RBF_kernel(xt[i], x[j])
        tmp.append(k_linear + k_RBF)
    K_test.append(tmp)
#print(K_test)
print("K_test has already been generated.")

model = svm_train(y, K, '-t 4 -b 1 -s 0')
p_label, p_acc, p_val = svm_predict(yt, K_test, model)

# toy testing
# beta = 0.2
# K = []
# train_count = 400
# for i in range(801, 1201):
#     tmp = []
#     tmp.append(i-801+1)
#     for j in range(801, 1201):
#         k_linear = linear_kernel(x[i], x[j])
#         k_RBF = RBF_kernel(x[i], x[j])
#         tmp.append(k_linear + k_RBF)
#         #tmp.append(k_linear)
#     tmp[i-801+1] += beta
#     K.append(tmp)
# #print(K)
# print("K has already been generated.")

# K_test = []
# test_count = 200
# for i in range(401, 601):
#     tmp = []
#     tmp.append(i-401+1)
#     for j in range(801, 1201):
#         k_linear = linear_kernel(xt[i], x[j])
#         k_RBF = RBF_kernel(xt[i], x[j])
#         tmp.append(k_linear + k_RBF)
#         #tmp.append(k_linear)
#     K_test.append(tmp)
# #print(K_test)
# print("K_test has already been generated.")

# model = svm_train(y[801:1201], K, '-t 4 -b 1 -s 0')
# p_label, p_acc, p_val = svm_predict(yt[401:601], K_test, model)
```

preprocessing

- We first load the data files and make them the format we need
- I write functions about linear kernel and RBF kernel both accept two pictures as two parameters, respectively, and both will return a scalar value

main

- I define a user-defined kernel K, K_test
- K is a 5000*5001 matrix
  - it's simply the kernel of each pair of data points in training data
  - the first column represents the number of the row(1,2,3,..., 2500)
  - the rest of the matrix is simply the kernel k(xi, yj) $\forall i, j = 1, 2, \ldots, 5000$
  - The matrix K looks like below
  - [1, k(x1,x1), k(x1,x2), ..., k(x1,x5000)]
  - [2, k(x2,x1), k(x2,x2), ..., k(x2,x5000)]
  - ...
  - [4999, k(x4999,x1), k(x4999,x2), ..., k(x4999, x5000)]
  - [5000, k(x5000,x1), k(x5000,x2), ..., k(x5000, x5000)]
- x: x-value in training data

- each entry in matrix K is linear_kernel of (xi, xj) plus RBF_kernel of (xi, xj)
- for those entries on the diagonal, we add a noise beta=0.2
- K_test is a 2500*5001 matrix
  - it's simply the kernel of each pair of data points between test data and training data
  - the first column represents the number of the row(1,2,3,..., 5000)
  - the rest of the matrix is simply the kernel k(xi, yj)
    $\forall i = 1, 2, \ldots, 2500, j = 1, 2, \ldots, 5000$
  - The matrix K_test looks like below
  - [1, k(x1*,x1), k(x1*,x2), ..., k(x1*,x5000)]
  - [2, k(x2*,x1), k(x2*,x2), ..., k(x2*,x5000)]
  - ...
  - [2499, k(x2499*,x1), k(x2499*,x2), ..., k(x2499*, x5000)]
  - [2500, k(x2500*,x1), k(x2500*,x2), ..., k(x2500*, x5000)]
- x: x-value in training data
- x*: x-value in test data
- each entry in matrix K_test is linear_kernel of (xi*, xj) plus RBF_kernel of (xi*, xj)
- model = svm_train(y, K, '-t 4 -b 1 -s 0')
  - we pass in y(training label)
  - we pass in K, the kernel matrix instead of x (training data)
  - -t 4 means to use precomputed kernel
  - -b 1 means to compute accuracy
  - -s 0 means C-SVC
- p_label, p_acc, p_val = svm_predict(yt, K_test, model)
  - yt(test label)
  - K_test, the kernel matrix instead of test data
  - use the trained model to make prediction
-

```
In [23]: import numpy as np
         from libsvm.svmutil import *
         from libsvm.svm import *

         Y_train = open('Y_train.csv', 'r')
         X_train = open('X_train.csv', 'r')
         Y_test = open('Y_test.csv', 'r')
         X_test = open('X_test.csv', 'r')

         y = []
         for line in Y_train.readlines():
             line = int(line)
             y.append(line)
         #print(y)

         x = []
         for line in X_train.readlines():
             tmp = line.split(',')
             elem = []
             for pixel in tmp:
                 pixel = float(pixel)
                 elem.append(pixel)
             x.append(elem)
         #print(x[0])

         yt = []
         for line in Y_test.readlines():
             line = int(line)
             yt.append(line)

         xt = []
```

```
yt = []
for line in Y_test.readlines():
    line = int(line)
    yt.append(line)

xt = []
for line in X_test.readlines():
    tmp = line.split(',')
    elem = []
    for pixel in tmp:
        pixel = float(pixel)
        elem.append(pixel)
    xt.append(elem)

## # q2_3
def linear_kernel(xa, xb):
    xa = np.array(xa)
    xb = np.array(xb)
    #print("xa {}".format(xa))
    #print(xa.T @ xb)
    return xa.T @ xb

def RBF_kernel(xa, xb):
    gamma = 1/784
    xa_subtract_xb = [xa[i] - xb[i] for i in range(len(xa))]
    xa_subtract_xb = np.array(xa_subtract_xb)
    #xa_subtract_xb = np.array([a-b for (a, b) in zip(xa, xb)])
    #print("\nxa {}\nxb {}\n xa_subtract_xb {}".format(xa, xb, xa_subtract_xb))
    #print(xa_subtract_xb.T @ xa_subtract_xb/2/sigma/sigma)
    #print(xa_subtract_xb.T @ xa_subtract_xb)
    return np.exp(-1*gamma*(xa_subtract_xb.T @ xa_subtract_xb))

#print(linear_kernel([7,8,9], [1,2,3]))
```

```
def RBF_kernel(xa, xb):
    gamma = 1/784
    xa_subtract_xb = [xa[i] - xb[i] for i in range(len(xa))]
    xa_subtract_xb = np.array(xa_subtract_xb)
    #xa_subtract_xb = np.array([a-b for (a, b) in zip(xa, xb)])
    #print("\nxa {}\nxb {}\n xa_subtract_xb {}".format(xa, xb, xa_subtract_xb))
    #print(xa_subtract_xb.T @ xa_subtract_xb/2/sigma/sigma)
    #print(xa_subtract_xb.T @ xa_subtract_xb)
    return np.exp(-1*gamma*(xa_subtract_xb.T @ xa_subtract_xb))

#print(linear_kernel([7,8,9], [1,2,3]))
#print(RBF_kernel([7,8,9], [1,2,3]))

beta = 0.2
K = []
train_count = len(x)
for i in range(train_count):
    tmp = []
    tmp.append(i+1)
    for j in range(train_count):
        k_linear = linear_kernel(x[i], x[j])
        k_RBF = RBF_kernel(x[i], x[j])
        tmp.append(k_linear + k_RBF)
    tmp[i+1] += beta
    K.append(tmp)
#print(K)
print("K has already been generated.")

K_test = []
test_count = len(xt)
for i in range(test_count):
    tmp = []
    tmp.append(i+1)
```

```
beta = 0.2
K = []
train_count = len(x)
for i in range(train_count):
    tmp = []
    tmp.append(i+1)
    for j in range(train_count):
        k_linear = linear_kernel(x[i], x[j])
        k_RBF = RBF_kernel(x[i], x[j])
        tmp.append(k_linear + k_RBF)
    tmp[i+1] += beta
    K.append(tmp)
#print(K)
print("K has already been generated.")

K_test = []
test_count = len(xt)
for i in range(test_count):
    tmp = []
    tmp.append(i+1)
    for j in range(train_count):
        k_linear = linear_kernel(xt[i], x[j])
        k_RBF = RBF_kernel(xt[i], x[j])
        tmp.append(k_linear + k_RBF)
    K_test.append(tmp)
#print(K_test)
print("K_test has already been generated.")

model = svm_train(y, K, '-t 4 -b 1 -s 0')
p_label, p_acc, p_val = svm_predict(yt, K_test, model)

# toy testing
# beta = 0.2
```

```
# toy testing
# beta = 0.2
# K = []
# train_count = 400
# for i in range(801, 1201):
#     tmp = []
#     tmp.append(i-801+1)
#     for j in range(801, 1201):
#         k_linear = linear_kernel(x[i], x[j])
#         k_RBF = RBF_kernel(x[i], x[j])
#         tmp.append(k_linear + k_RBF)
#         #tmp.append(k_linear)
#     tmp[i-801+1] += beta
#     K.append(tmp)
# #print(K)
# print("K has already been generated.")

# K_test = []
# test_count = 200
# for i in range(401, 601):
#     tmp = []
#     tmp.append(i-401+1)
#     for j in range(801, 1201):
#         k_linear = linear_kernel(xt[i], x[j])
#         k_RBF = RBF_kernel(xt[i], x[j])
#         tmp.append(k_linear + k_RBF)
#         #tmp.append(k_linear)
#     K_test.append(tmp)
# #print(K_test)
# print("K_test has already been generated.")

# model = svm_train(y[801:1201], K, '-t 4 -b 1 -s 0')
# p_label, p_acc, p_val = svm_predict(yt[401:601], K_test, model)
```

```
#     tmp.append(i-801+1)
#     for j in range(801, 1201):
#         k_linear = linear_kernel(x[i], x[j])
#         k_RBF = RBF_kernel(x[i], x[j])
#         tmp.append(k_linear + k_RBF)
#         #tmp.append(k_linear)
#     tmp[i-801+1] += beta
#     K.append(tmp)
# #print(K)
# print("K has already been generated.")

# K_test = []
# test_count = 200
# for i in range(401, 601):
#     tmp = []
#     tmp.append(i-401+1)
#     for j in range(801, 1201):
#         k_linear = linear_kernel(xt[i], x[j])
#         k_RBF = RBF_kernel(xt[i], x[j])
#         tmp.append(k_linear + k_RBF)
#         #tmp.append(k_linear)
#     K_test.append(tmp)
# #print(K_test)
# print("K_test has already been generated.")

# model = svm_train(y[801:1201], K, '-t 4 -b 1 -s 0')
# p_label, p_acc, p_val = svm_predict(yt[401:601], K_test, model)
```

```
K has already been generated.
K_test has already been generated.
Model supports probability estimates, but disabled in predicton.
Accuracy = 95.24% (2381/2500) (classification)
```

- we got accuracy 95.24%, which is worse than just using linear_kernel(96%) or just using RBF_kernel(98.52%)
- The program runs for 3.5hours, I'll find a way to speed it up afterwards.

reference

- https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf (https://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf)
- https://www.itread01.com/content/1496679627.html (https://www.itread01.com/content/1496679627.html)
- https://www.ycc.idv.tw/ml-course-techniques_2.html (https://www.ycc.idv.tw/ml-course-techniques_2.html)