Homework 4 程式題

12. (d) 3.0
13. (c) 0.0
14. (d) [1, 151, 83, 17, 4]
15. (c) 0.16947656250000007
16. (b) 0.14893750000000022
17. (a) 0.124
18. (c) 0.0
19. (e) 960
20. (a) 1

For 12~20 every code have the code below

```
import numpy as np
import random
import math
from liblinear.liblinearutil import *

d = 10
Q = 4
C_list = [0.0000005, 0.0005, 0.5, 500, 500000]    #C = 1 / 2 * lambda for L2
#C_list = [0.000001, 0.001, 1, 1000, 1000000]    #C = 1 / lambda for L1

filename1 = 'train.txt'
train_N = 200
X = np.ones([train_N, d+1])
y = np.zeros(train_N)
with open(filename1) as file:
    a = 0
    for line in file:
        line = line.strip().split()
        for i in range(d):
            X[a][i+1] = line[i]
        X[a][0] = 1
        y[a] = line[d]
        a += 1
file.close()

filename2 = 'test.txt'
test_N = 500
X_test = np.ones([test_N, d+1])
y_test = np.zeros(test_N)
with open(filename2) as file:
    a = 0
    for line in file:
        line = line.strip().split()
        for i in range(d):
            X_test[a][i+1] = line[i]
            X_test[a][0] = 1
            y_test[a] = line[d]
            a += 1
file.close()

#code below is for transform
X_tran = np.ones([train_N, 1001])
for i in range(train_N):
    cnt = 1
    for j in range(d):
        X_tran[i][cnt] = X[i][j+1]
        cnt += 1
    x_tmp1 = X_tran[i][1:11]
    x_tmp_index = np.arange(d+1)
    x_tmp_index_n = np.arange(d+1)
    for j in range(d):
        x_tmp2 = X_tran[i][j+1] * x_tmp1[x_tmp_index[j]:]
        x_tmp_index_n[j+1] = np.size(x_tmp2) + x_tmp_index_n[j]
        for k in range(np.size(x_tmp2)):
            X_tran[i][cnt] = x_tmp2[k]
            cnt += 1
    x_tmp1 = X_tran[i][11:66]
    x_tmp_index_nn = np.arange(d+1)
    for j in range(d):
        x_tmp2 = X_tran[i][j+1] * x_tmp1[x_tmp_index_n[j]:]
        x_tmp_index_nn[j+1] = np.size(x_tmp2) + x_tmp_index_nn[j]
        for k in range(np.size(x_tmp2)):
            X_tran[i][cnt] = x_tmp2[k]
            cnt += 1
    x_tmp1 = X_tran[i][66:286]
    x_tmp_index_nnn = np.arange(d+1)
    for j in range(d):
        x_tmp2 = X_tran[i][j+1] * x_tmp1[x_tmp_index_nn[j]:]
        x_tmp_index_nnn[j+1] = np.size(x_tmp2) + x_tmp_index_nnn[j]
        for k in range(np.size(x_tmp2)):
            X_tran[i][cnt] = x_tmp2[k]
            cnt += 1

X_test_tran = np.ones([test_N, 1001])
for i in range(test_N):
    cnt = 1
    for j in range(d):
        X_test_tran[i][cnt] = X_test[i][j+1]
        cnt += 1
    x_tmp1 = X_test_tran[i][1:11]
    x_tmp_index = np.arange(d+1)
    x_tmp_index_n = np.arange(d+1)
    for j in range(d):
        x_tmp2 = X_test_tran[i][j+1] * x_tmp1[x_tmp_index[j]:]
        x_tmp_index_n[j+1] = np.size(x_tmp2) + x_tmp_index_n[j]
        for k in range(np.size(x_tmp2)):
            X_test_tran[i][cnt] = x_tmp2[k]
            cnt += 1
    x_tmp1 = X_test_tran[i][11:66]
    x_tmp_index_nn = np.arange(d+1)
    for j in range(d):
        x_tmp2 = X_test_tran[i][j+1] * x_tmp1[x_tmp_index_n[j]:]
        x_tmp_index_nn[j+1] = np.size(x_tmp2) + x_tmp_index_nn[j]
        for k in range(np.size(x_tmp2)):
            X_test_tran[i][cnt] = x_tmp2[k]
            cnt += 1
    x_tmp1 = X_test_tran[i][66:286]
    x_tmp_index_nnn = np.arange(d+1)
    for j in range(d):
        x_tmp2 = X_test_tran[i][j+1] * x_tmp1[x_tmp_index_nn[j]:]
        x_tmp_index_nnn[j+1] = np.size(x_tmp2) + x_tmp_index_nnn[j]
        for k in range(np.size(x_tmp2)):
            X_test_tran[i][cnt] = x_tmp2[k]
            cnt += 1
```

Q12:

```
E_out = 1
index = 0
for i in range(5):
    prob = problem(y, X_tran)
    param = parameter(f'-s 0 -c {(C_list[i])} -e 0.000001 -q')
    model_ptr = liblinear.train(prob, param)
    model_ = toPyModel(model_ptr)
    [W_out, b_out] = model_.get_decfun()

    E_out_tmp = 0
    for j in range(test_N):
        if np.sign(np.dot(W_out, X_test_tran[j])) != y_test[j]:
            E_out_tmp += 1
    E_out_tmp = E_out_tmp / test_N
    if E_out_tmp < E_out:
        E_out = E_out_tmp
        index = i

print(math.log10(1/(2*C_list[index])))
```

Q13:

```
E_in = 1
index = 0
for i in range(5):
    prob = problem(y, X_tran)
    param = parameter(f'-s 0 -c {(C_list[i])} -e 0.000001 -q')
    model_ptr = liblinear.train(prob, param)
    model_ = toPyModel(model_ptr)
    [W_in, b_in] = model_.get_decfun()
    E_in_tmp = 0
    for j in range(train_N):
        if np.sign(np.dot(W_in, X_tran[j])) != y[j]:
            E_in_tmp += 1
    E_in_tmp = E_in_tmp / train_N
    if E_in_tmp < E_in:
        E_in = E_in_tmp
        index = i

print(math.log10(1/(2*C_list[index])))
```

Q14:

```
cnt_list = [0,0,0,0,0]
for i in range(256):
    list1 = random.sample(range(train_N), train_N)
    D_train = np.zeros([120, 1001])
    D_y_train = np.zeros(120)
    D_val = np.zeros([80, 1001])
    D_y_val = np.zeros(80)
    for j in range(120):
        D_train[j] = X_tran[list1[j]]
        D_y_train[j] = y[list1[j]]
    for j in range(80):
        D_val[j] = X_tran[list1[120+j]]
        D_y_val[j] = y[list1[120+j]]

    E_val = 1
    index = 0
    for j in range(5):
        prob = problem(D_y_train, D_train)
        param = parameter(f'-s 0 -c {(C_list[j])} -e 0.000001 -q')
        model_ptr = liblinear.train(prob, param)
        model_ = toPyModel(model_ptr)
        [W_sam, b_sam] = model_.get_decfun()

        E_val_tmp = 0
        for k in range(80):
            if np.sign(np.dot(W_sam, D_val[k])) != D_y_val[k]:
                E_val_tmp += 1
        E_val_tmp = E_val_tmp / 80
        if(E_val_tmp < E_val):
            E_val = E_val_tmp
            index = j
    cnt_list[index] += 1

print(cnt_list)
```

Q15:

```
E_out_avg = 0
for i in range(256):
    list1 = random.sample(range(train_N), train_N)
    D_train = np.zeros([120, 1001])
    D_y_train = np.zeros(120)
    D_val = np.zeros([80, 1001])
    D_y_val = np.zeros(80)
    for j in range(120):
        D_train[j] = X_tran[list1[j]]
        D_y_train[j] = y[list1[j]]
    for j in range(80):
        D_val[j] = X_tran[list1[120+j]]
        D_y_val[j] = y[list1[120+j]]

    E_val = 1
    w_b = 0
    for j in range(5):
        prob = problem(D_y_train, D_train)
        param = parameter(f'-s 0 -c {(C_list[j])} -e 0.000001 -q')
        model_ptr = liblinear.train(prob, param)
        model_ = toPyModel(model_ptr)
        [W_sam, b_sam] = model_.get_decfun()

        E_val_tmp = 0
        for k in range(80):
            if np.sign(np.dot(W_sam, D_val[k])) != D_y_val[k]:
                E_val_tmp += 1
        E_val_tmp = E_val_tmp / 80
        if(E_val_tmp < E_val):
            E_val = E_val_tmp
            w_b = W_sam
    E_out = 0
    for i in range(test_N):
        if np.sign(np.dot(w_b, X_test_tran[i])) != y_test[i]:
            E_out += 1
    E_out = E_out / test_N
    E_out_avg += E_out
E_out_avg = E_out_avg / 256
print(E_out_avg)
```

Q16:

```
E_out_avg = 0
for i in range(256):
    list1 = random.sample(range(train_N), train_N)
    D_train = np.zeros([120, 1001])
    D_y_train = np.zeros(120)
    D_val = np.zeros([80, 1001])
    D_y_val = np.zeros(80)
    for j in range(120):
        D_train[j] = X_tran[list1[j]]
```

```
            D_y_train[j] = y[list1[j]]
        for j in range(80):
            D_val[j] = X_tran[list1[120+j]]
            D_y_val[j] = y[list1[120+j]]

        E_val = 1
        index = 0
        for j in range(5):
            prob = problem(D_y_train, D_train)
            param = parameter(f'-s 0 -c {(C_list[j])} -e 0.000001 -q')
            model_ptr = liblinear.train(prob, param)
            model_ = toPyModel(model_ptr)
            [W_sam, b_sam] = model_.get_decfun()

            E_val_tmp = 0
            for k in range(80):
                if np.sign(np.dot(W_sam, D_val[k])) != D_y_val[k]:
                    E_val_tmp += 1
            E_val_tmp = E_val_tmp / 80
            if(E_val_tmp < E_val):
                index = j
                E_val = E_val_tmp
        prob = problem(y, X_tran)
        param = parameter(f'-s 0 -c {(C_list[index])} -e 0.000001 -q')
        model_ptr = liblinear.train(prob, param)
        model_ = toPyModel(model_ptr)
        [w_b, b_b] = model_.get_decfun()

        E_out = 0
        for i in range(test_N):
            if np.sign(np.dot(w_b, X_test_tran[i])) != y_test[i]:
                E_out += 1
        E_out = E_out / test_N
        E_out_avg += E_out
    E_out_avg = E_out_avg / 256
    print(E_out_avg)
```

Q17:

```
    E_cv_sum = 0
    for i in range(256):
        list1 = list(range(train_N))
        random.shuffle(list1)
        D_fold = np.zeros([5, 40, 1001])
        D_y = np.zeros([5, 40])
        for j in range(5):
            for k in range(40):
                D_fold[j][k] = X_tran[list1[40*j+k]]
                D_y[j][k] = y[list1[40*j+k]]
        E_cv_avg = 0
        D_train = np.zeros([160, 1001])
        D_y_train = np.zeros(160)
        list2 = [[1,2,3,4],[0,2,3,4],[0,1,3,4],[0,1,2,4],[0,1,2,3]]
        for j in range(5):
            E_cv = 1
            for k in range(5):
                for q in range(4):
                    for p in range(40):
                        D_train[p + 40*q] = D_fold[list2[j][q]][p]
                        D_y_train[p + 40*q] = D_y[list2[j][q]][p]
                prob = problem(D_y_train, D_train)
                param = parameter(f'-s 0 -c {C_list[k]} -e 0.000001 -q')
                m = train(prob, param)
                p_labs, p_acc, p_vals = predict(D_y[j], D_fold[j], m,'-q')
                E_cv_tmp = 1 - p_acc[0]/100
                if(E_cv_tmp < E_cv):
                    E_cv = E_cv_tmp
            E_cv_avg += E_cv
        E_cv_sum += E_cv_avg / 5
    E_cv_sum = E_cv_sum / 256
    print(E_cv_sum)
```

Q18:

```
E_out = 1
index = 0
for i in range(5):
    prob = problem(y, X_tran)
    param = parameter(f'-s 6 -c {(C_list[i])} -e 0.000001 -q')
    model_ptr = liblinear.train(prob, param)
    model_ = toPyModel(model_ptr)
    [W_out, b_out] = model_.get_decfun()

    E_out_tmp = 0
    for j in range(test_N):
        if np.sign(np.dot(W_out, X_test_tran[j])) != y_test[j]:
            E_out_tmp += 1
    E_out_tmp = E_out_tmp / test_N
    if E_out_tmp < E_out:
        E_out = E_out_tmp
        index = i

print(math.log10(1/C_list[index]))
```

Q19:

```
prob = problem(y, X_tran)
param = parameter(f'-s 6 -c {(C_list[2])} -e 0.000001 -q')
model_ptr = liblinear.train(prob, param)
model_ = toPyModel(model_ptr)
```

```
[W_out, b_out] = model_.get_decfun()

sparse_cnt = 0
for i in range(1001):
    if abs(W_out[i]) <= 0.000001:
        sparse_cnt += 1
print(sparse_cnt)
```

Q20:

```
prob = problem(y, X_tran)
param = parameter(f'-s 0 -c {(C_list[1])} -e 0.000001 -q')
model_ptr = liblinear.train(prob, param)
model_ = toPyModel(model_ptr)
[W_out, b_out] = model_.get_decfun()

sparse_cnt = 0
for i in range(1001):
    if abs(W_out[i]) <= 0.000001:
        sparse_cnt += 1
print(sparse_cnt)
```