1.

$$\frac{2}{N}\sum_{n=1}^{N}(w^{*}\cdot x_n - y_n)\cdot x_n + \frac{2\lambda}{N}w^{*} = 0$$

$$\Rightarrow w^{*}\cdot\sum_{n=1}^{N}x_n^{2} - \sum_{n=1}^{N}x_n\cdot y_n + \lambda\cdot w^{*} = 0 \Rightarrow w^{*} = \frac{\sum_{n=1}^{N}x_n\cdot y_n}{\sum_{n=1}^{N}x_n^{2}+\lambda}$$

$$C = (w^{*})^{2} = \left(\frac{\sum_{n=1}^{N}x_n\cdot y_n}{\sum_{n=1}^{N}x_n^{2}+\lambda}\right)^{2}$$

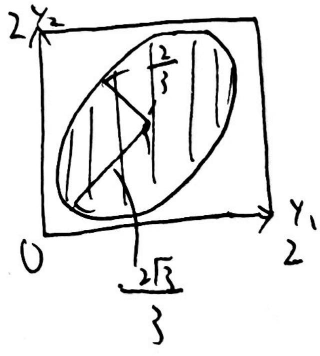
2. $\Phi(x) = \Gamma^{-1}\cdot x \Rightarrow x = \Gamma\cdot\Phi(x)$

$$\frac{1}{N}\sum_{n=1}^{N}(w^{T}x_n - y_n)^{2} = \frac{1}{N}\sum_{n=1}^{N}(w^{T}\cdot\Gamma\cdot\Phi(x_n) - y_n)^{2} \Rightarrow \tilde{w}^{T} = w^{T}\cdot\Gamma$$

$$\Omega(w) = (\tilde{w}^{T}\cdot\tilde{w}) = (w^{T}\cdot\Gamma\cdot\Gamma^{T}\cdot w) = w^{T}\cdot\Gamma^{2}\cdot w$$

4. $$E_{loocv} = \frac{1}{N}\sum_{i=1}^{N}e_i = \frac{1}{3}\left(\left(\frac{y_2+y_3}{2}-y_1\right)^{2} + \left(\frac{y_1+y_3}{2}-y_2\right)^{2} + \left(\frac{y_1+y_2}{2}-y_3\right)^{2}\right) \le \frac{1}{3}$$

$$\Rightarrow y_1^{2}+y_2^{2}-y_1-y_2-y_1y_2+1 \le \frac{2}{3}$$



$$P\left(E_{loocv} \le \frac{1}{3}\right) = \frac{\frac{2}{3}\cdot\frac{2\sqrt{3}}{3}\cdot\pi}{4} = \frac{\sqrt{3}}{9}\pi = \frac{\pi}{3\sqrt{3}}$$

6. Since when we leave one data out it will always predict wrong $e_i = 1$

$$E_{loocv} = \frac{1}{2N}\cdot\sum_{i=1}^{2N}\cdot e_i = \frac{1}{2N}\cdot 2N = 1$$

3. $D_{KL}(P_u||P_h) = \frac{1}{2}\cdot\ln\frac{\frac{1}{2}}{h(x)} + \frac{1}{2}\cdot\ln\frac{\frac{1}{2}}{1-h(x)} = \frac{1}{2}(\ln(1+e^{-w^Tx}) + \ln(1+e^{w^Tx})) - \ln 2$

For $y_n \doteq +1$: (similar things when $y=-1$)

① $\underline{\text{err}_{\text{smooth}}(w,x_n,+1)} = (\frac{2-\alpha}{2})\cdot\ln(1+e^{-w^Tx}) + \frac{\alpha}{2}\ln(1+e^{w^Tx})$

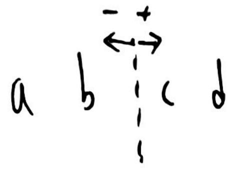
let $\Omega(w,x_n) = D_{KL}(P_u||P_h)$:

② $\underline{\text{err}(w,x_n,+1) + \lambda\cdot\Omega(w,x_n)} = \ln(1+e^{-w^Tx}) + \frac{\alpha}{1-\alpha}\cdot\frac{1}{2}(\ln(1+e^{-w^Tx}) + \ln(1+e^{w^Tx})) - \frac{\alpha\cdot\ln 2}{1-\alpha}$

$$= (\frac{2-\alpha}{2-2\alpha})\cdot\ln(1+e^{-w^Tx}) + (\frac{\alpha}{2-2\alpha})\cdot\ln(1+e^{w^Tx}) + \text{Constant}$$

Solving minimize problem of ① is equivalent to ②, because constant can be ignore when finding the gradient, so the difference between the gradient of ① and ② is mutiple $\frac{1}{1-\alpha}$, since the min happens when gradient $=0$ we can also get rid of the mutiple, however in pratical α is usually a small number <1.

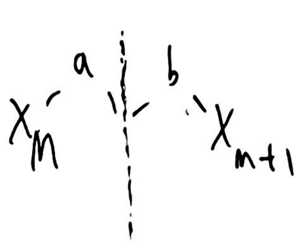
7. for $N \geq 4$ the upper bound of E_{loocv} happens when, $-1 \leq a < b < 0 < c < d \leq 1$



, $b > \frac{a+c}{2}$ and $c < \frac{b+d}{2}$, this will make e_b and $e_c = 1$

and all other $e_i = 0$, so $E_{loocv} = \frac{1}{N}\sum_{i=1}^{N} e_i \leq \frac{1}{N} \cdot 2$

8. To get the largest margin



, $a = b = \text{margin}_{max}$

$\text{margin}_{max} = \frac{1}{2} \cdot (X_{m+1} - X_m)$

9. for $w = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$, $\begin{cases} 4\beta + b \geq 1 \\ -(2\alpha + b) \geq 1126 \\ -\alpha + b \geq 1 \\ b \geq 1 \end{cases} \Longrightarrow \begin{cases} \beta \geq \frac{1-b}{4} \\ \alpha \leq \frac{-1126-b}{2} \\ b \geq 1 \end{cases}$

$\xRightarrow{\min\limits_{w,b} \frac{1}{2} w^T w} \min\limits_{\alpha,\beta,b} \alpha^2 + \beta^2 \Rightarrow b = 1,\ \alpha = \frac{-1127}{2},\ \beta = 0$

10. We want $E_{in}(\hat{h}) = \frac{1}{N}\sum_{i=1}^{N} [\![\hat{h}(x_i) \neq y_i]\!] = 0$

$\Rightarrow$ for all $i \in N$, $\hat{h}(x_i) = y_i \Rightarrow \text{sign}\left(\sum_{n \in N} y_n \cdot e^{-r\|x_n - x_i\|^2}\right) = y_i$

$\Rightarrow \text{sign}\left(\underbrace{y_i \cdot e^{-r\|x_i - x_i\|^2}}_{y_i} + \sum_{n \in N, n \neq i} y_n \cdot e^{-r\|x_n - x_i\|^2}\right) = y_i$, so $|y_i| > \left|\sum_{n \in N, n \neq i} y_n \cdot e^{-r\|x_n - x_i\|^2}\right|$

for y_i to dominate the sign

$\Rightarrow 1 > \sum_{n \in N, n \neq i} e^{-r\|x_n - x_i\|^2} \underbrace{\geq}_{\text{since } \|x_n - x_m\| > \epsilon \ \forall\, n \neq m} \sum_{n \in N, n \neq i} e^{-r\epsilon^2} = (N-1) \cdot e^{-r\epsilon^2} \Rightarrow e^{-r\epsilon^2} < \frac{1}{N-1}$

$\Rightarrow -r \cdot \epsilon^2 < -\ln(N-1) \Rightarrow r > \frac{\ln(N-1)}{\epsilon^2}$

11. $\|\phi(x) - \phi(x')\|^2 = K(x,x) - 2K(x,x') + K(x',x') = 2 - 2\underbrace{K(x,x')}_{>0}$

$\Rightarrow \|\phi(x) - \phi(x')\|^2 < 2 \Rightarrow \|\phi(x) - \phi(x')\| < \sqrt{2} < 1{,}5$

5. $\underset{D_{val} \sim P^k}{\text{Var}}[E_{val}(h)] = \underset{D_{val} \sim P^k}{\text{Var}}\left[\frac{1}{K}\sum_{i=1}^{K} err(h(x_i), y_i)\right] = \frac{1}{K^2} \underset{D_{val} \sim P^k}{\text{Var}}\left[\sum_{i=1}^{K} err(h(x_i), y_i)\right]$

Since (x,y) in D_{val} is i.i.d we can take the summation off $\Rightarrow \frac{1}{K^2} \cdot K \cdot \underset{(x,y) \sim P}{\text{Var}}[err(h(x), y)] = \frac{1}{K} \underset{(x,y) \sim P}{\text{Var}}[err(h(x), y)]$

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)
```