Machine Learning HW5. 蔡家豪

1. if we were to correcly classify.

we need $y_n(w^T x_n + b) \geq 1 - \xi_n$ , $\xi_n \geq 0$ for $n = 1 \cdots N$

which means when $y_n = 1$, we need $0 \leq \xi_n < 1$ to make $y_n(w^T x_n + b) \geq 1 - \xi_n$

when we consider $y_n = -1$ and $w^T x_n + b > 0$ , this is obviously a missclassified case, we will have $y_n(w^T x_n + b) < 0$

if we still want to follow the constraint, we need $\xi_n > 1$ in oder to make $1 - \xi_n < 0$ , which will make $y_n(w^T x_n + b) \geq 1 - \xi_n$

this relation will make sure $\sum_{n \in \text{missclassified}}^{N} \xi_n > \sum_{n \in \text{missclassified}}^{N} 1$

also, when we comes to the correcly classify, we know $\sum_{n \in \text{correct}} \xi_n \geq 0$ , since we have $\xi_n \geq 0$

we can derive the relation $\sum_{n=1}^{N} \xi_n = \sum_{n \in \text{missclassified}}^{N} \xi_n + \sum_{n \in \text{correct}}^{N} \xi_n > \sum_{n \in \text{missclassified}}^{N} 1$

which means $\sum_{n=1}^{N} \xi_n^*$ is an upper bound of missclassified sample $+ 0$

also, cuz we have the relation $\sum_{n \in \text{missclassified}}^{N} \xi_n > \sum_{n \in \text{missclassified}}^{N} 1$

no matter we take $\lfloor \ , \ \rfloor$, we can still make sure $\lfloor 1 \rfloor$ and $\lfloor \lfloor \ \rfloor \rfloor$ but $\frac{1}{2}$ cannot, so $\log(1+\xi_n^*)$, $\sum_{n=1}^{N} \lfloor \xi_n^* \rfloor$ and $\sum_{n=1}^{N} \lfloor \xi_n^* \rfloor$ both upper bounds too.

Ans [d] 4 #

$\Sigma . \lfloor \xi_n \rfloor > \lfloor 1 \rfloor = 1$

$\Sigma \log_2(1+\xi_n^*) > \log_2(1+1) = 1$

$\Sigma \lfloor \xi_n^* \rfloor > \lfloor 1 \rfloor = 1$

2. We already know $\forall n$, $\alpha_n^* = C$

by complementary slackness

$$\forall n, \alpha_n (1 - \xi_n - y_n(w^T x_n + b)) = 0$$

$$\Rightarrow 1 - \xi_n - y_n(w^T x_n + b) = 0$$

$$\Rightarrow y_n b = 1 - \xi_n - y_n w^T x_n = 1 - \xi_n - y_n \sum_{i=1}^{N} \alpha_i y_i x_i^T x_n$$

we take $y_n = -1$ to obtain the lower bound

$$\Rightarrow -b = 1 - \xi_n + \sum_{m=1}^{N} \alpha_m y_m x_m^T x_n$$

$$\Rightarrow b = -1 + \xi_n - \sum_{m=1}^{N} \alpha_m y_m x_m^T x_n \quad \text{cuz } \xi_n \geq 0.$$

$$\Rightarrow b = -1 + \xi_n - \sum_{m=1}^{N} \alpha_m y_m x_m^T x_n \geq -1 - \sum_{m=1}^{N} \alpha_m y_m x_n^T x_n$$

cuz we want the lower bound of $b$, so we need to take max

we have

$$\Rightarrow b \geq \max_{n: y_n < 0} \left(-1 - \sum_{m=1}^{N} y_m \alpha_m^* K(x_n, x_m)\right) \quad \text{[d]}$$

3.

$$\min \frac{1}{2} w^T w + C \sum \xi_n^2 \quad \text{s.t. } \forall n \; y_n(w^T z_n + b) \geq 1 - \xi_n.$$

$$Lp(w, b, \xi, \alpha) = \frac{1}{2}\|w\|^2 + C \sum \xi_n^2 - \sum_{i=1}^{N} \alpha_i (y_i (y_i x_i w + b) - 1 + \xi_i)$$

$$\Rightarrow \frac{\partial Lp(\,)}{\partial \xi_i} = 0 = 2C \sum \xi_i - \sum_{i=1}^{n} \alpha_i = 0$$

$$\Rightarrow \sum \xi_i = \frac{1}{2C} \sum \alpha_i$$

$$\Rightarrow \xi^* = \frac{1}{2C} \alpha^* \quad \text{[a]}$$

4.


We have $K_{ds}(x, x') = g_{m,1,2L+1}(x) g_{m,1,2L+1}(x') + \cdots + g_{-1,d,2R-1}(x) g_{-1,d,2R-1}(x')$

also, $g(x) g(x') = s \cdot \text{sign}(x-\theta) \cdot s \cdot \text{sign}(x'-\theta)$, we know $s$ is one the same $\Rightarrow s^2 = 1$

$$= \text{sign}(x-\theta) \text{sign}(x'-\theta)$$

when $x-\theta > 0$, $x'-\theta < 0 \Rightarrow g(x) g(x') = -1$.

$\Rightarrow$ which means when $\theta \in [x+1, x'-1] \Rightarrow g(x) g(x') = -1$

other case , $g(x)\,g(x') = 1$

so , we will have

$$2L+1 \sim 2R-1 \quad [\ \underbrace{\frac{(2R-1)-(2L+1)}{2}}_{\text{only odd}} + 1\ ] \text{ term} = R-L$$

also, we have d dimension and 2 direction $\Rightarrow 2 \times d\,(R-L)$

total term will be $2 \cdot d \cdot (R-L)$, if every term is $+1$

now consider the $-1$ case

if $x < \theta < x'$ , we have $\frac{|x-x'|}{2}$ of them $\theta$ (even only)

consider 2 direction we have $2 \times \frac{!}{=}(x-x'| = |x-x'|$

$\overset{!}{=} \sum [g(x)\,g(x')=1] - \sum [g(x)\,g(x')=-1] = \underbrace{2d(R-L) - 2|x-x'|}_{\text{no choice}}$ → can't $+1 \Rightarrow -1$ needs 2 $\quad$ [E]

$\leq$.

When we consider the "tighest upper bound", we only need $m+1$ to be wrong, which will become error. $\left((2M+N+1)/2 = M+1\right)$

So Ans should be [b] $\frac{1}{M+1} \sum_{t=1}^{2M+1} e_t$ .

6.

$N = 112$, select $N'$ from $N$

all different : $\binom{N}{1}\binom{N-1}{1} \cdots \binom{N-N'+1}{1}$

$$= \frac{N!}{(N-1)!} \cdot \frac{(N-1)!}{(N-2)!} \cdots \frac{(N-N'+1)!}{(N-N')!} = \frac{N!}{(N-N')!}$$

all possibility : $N^{N'}$

$$\Rightarrow \frac{N^{N'} - \frac{N!}{(N-N')!}}{N^{N'}} = 1 - \frac{N!}{(N-N')! \cdot N^{N'}} \geq 0.75$$

$\Rightarrow \frac{N!}{(N-N')! \cdot N^{N'}} \leq 0.25$ $\Rightarrow$ take ln and use program to calculate.

(a) 54 $\Rightarrow$ 0.275 $>$ 0.25 (X)

(b) 56 $\Rightarrow$ 0.2495 $<$ 0.25 (O) $\quad$ [b]

(c) 58 $\Rightarrow$ 0.2298 $<$ 0.25 (O)

(d) 60 $\Rightarrow$ 0.2021 $<$ 0.25 (O)

7.

$$\min_w E_{in}^u(w) = \frac{1}{N}\sum_{n=1}^{N} u_n (y_n - w^T x_n)^2.$$

$$\Rightarrow \min_w E_{in}^u(w) = \frac{1}{N}\sum_{n=1}^{N} \left(\sqrt{u_n}\, y_n - w^T \sqrt{u_n}\, x_n\right)^2.$$

$$\Rightarrow \underline{\hat{x}_n = \sqrt{u_n}\, x_n \quad \checkmark}$$
$$\underline{\hat{y}_n = \sqrt{u_n}\, y_n \quad \#}$$

[C] ✓

8. [a]

1st gini : $1 - \left(\frac{50}{50}\right)^2 - \left(\frac{0}{50}\right)^2 = 0$

2nd gini : $1 - \left(\frac{25}{50}\right)^2 - \left(\frac{25}{50}\right)^2 = \frac{1}{2}$

take weight $\frac{50}{100} \times 0 + \frac{50}{100} \times \frac{1}{2} = \underline{\frac{1}{4}}$ # $= 0.25$

[b]

1st : $1 - (0.8)^2 - (0.2)^2 = 0.32$

2nd : $1 - (0.75)^2 - (0.25)^2 = \frac{3}{8}$

$\frac{70}{100} \times 0.32 + \frac{30}{100} \times \frac{3}{8} = 0.3365$

(c)

1st : $1 - (0.7)^2 - (0.3)^2 = 0.42$

2nd : $1 - 1 = 0.$

$\frac{70}{100} \times 0.42 + \frac{10}{100} \times 0 = 0.378$

[d]

1st : $1 - (0.8)^2 - (0.2)^2 = 0.32$

2nd : $1 - (0.9)^2 - (0.1)^2 = 0.18$

$\frac{80}{100} \times 0.32 + \frac{20}{100} \times 0.18 = 0.292$

(e)

1st : $1 - (0.9)^2 - (0.1)^2 = 0.18.$

2nd : $1 - (0.9)^2 - (0.1)^2 = 0.18.$

$\frac{80}{100} \times 0.18 + \frac{20}{100} \times 0.18 = \underline{0.18}$ #

(e) ✓

9. $\epsilon_t = \frac{\sum_{n=1}^{N} u_n^{(t)} [y_n \neq g_t(x_n)]}{\sum_{n=1}^{N} u_n^{(t)}} \Rightarrow \sum_{n \in \text{incorrect}}^{N} u_n^{(t)}$

scaling factor $\diamond_t = \sqrt{\frac{1-\epsilon_t}{\epsilon_t}}$

$$\left( \frac{\sum_{n \in \text{correct}}^{N} u_n^{(t)}}{\sum_{n=1}^{N} u_n^{(t)}} \right) = \frac{1-\epsilon_t}{\frac{\sum_{n=1}^{N} u_n^{(t)} [y_n = g_t(x_n)]}{\sum_{n=1}^{N} u_n^{(t)}}}$$

$$\Rightarrow U^{(t+1)} = \sum_{n=1}^{N} u_n^{(t+1)}$$

$\Rightarrow$ if correct / incorrect $\times$ $\diamond_t^{(-y_n \cdot g_t(x_n))}$

$$= \sum_{n=1}^{N} u_n^{(t)} \cdot \diamond_t$$

$$= \sum_{n \in \text{correct}}^{N} u_n^{(t)} / \diamond_t + \sum_{n \in \text{incorrect}}^{N} u_n^{(t)} \cdot \diamond_t$$

$$= (1-\epsilon_t) \times \sum_{n=1}^{N} u_n^{(t)} \times \sqrt{\frac{\epsilon_t}{1-\epsilon_t}} + \epsilon_t \sum_{n=1}^{N} u_n^{(t)} \times \sqrt{\frac{1-\epsilon_t}{\epsilon_t}}$$

$$= \sum_{n=1}^{N} u_n^{(t)} \left( \sqrt{\epsilon_t(1-\epsilon_t)} + \sqrt{\epsilon_t(1-\epsilon_t)} \right)$$

$$= U^{(t)} \, 2\sqrt{\epsilon_t(1-\epsilon_t)}$$

if recursive ....

$\Rightarrow$ $2^T \prod_{t=1}^{T} \sqrt{\epsilon_t(1-\epsilon_t)}$ [d]

10. $\min_{\eta} \frac{1}{N} \sum_{n=1}^{N} (s_n + \eta g_t(x_n) - y_n)^2$.

$$\frac{\partial(\,)}{\partial \eta} = \frac{1}{N} \cdot 2 \cdot \sum_{n=1}^{N} (s_n + \eta g_t(x_n) - y_n) \times (g_t(x_n)) = 0.$$

$$\Rightarrow \sum_{n=1}^{N} (\boxed{s_n + \eta g_t(x_n)} - y_n) \times g_t(x_n) = 0$$

$s_{n+1}$

$$\Rightarrow \sum_{n=1}^{N} (s_{n+1} - y_n) \cdot g_t(x_n) = 0$$

$\Rightarrow$ (b)

Edit svm.cpp and recompile by libsvm FAQ
Q: How do I get the distance between a point and the hyperplane?

```
// calculate objective value
{
    double v = 0;
    int i;
    for(i=0;i<l;i++)
        v += alpha[i] * (G[i] + p[i]);
    si->obj = v/2;

    double alpha_sum = 0;
    for(i=0;i<l;i++)
        alpha_sum += alpha[i];
    printf("||w||^2 = %f\n", 2*(si->obj + alpha_sum));
}
```

```
# %%
import numpy as np
from libsvm.svmutil import *

C = 1
def label_transform(y, target_label):
    return list(map(lambda x: 2*x-1, np.array(y) == target_label))


y, x = svm_read_problem('./letter.scale.tr')
p11_y = label_transform(y, 1)
m = svm_train(p11_y, x, '-s 0 -t 0 -c 1')
```

```
.*||w||^2 = 39.811974

optimization finished, #iter = 1571
nu = 0.028677
obj = -281.199069, rho = 5.749916
nSV = 308, nBSV = 292
Total nSV = 308
```

sqrt(39.811974)
= 6.3096...

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

def label_transform(y, target_label):
    return list(map(lambda x: 2*x-1, np.array(y) == target_label))

y, x = svm_read_problem('./letter.scale.tr')

C = 1
Q = 2

ein_dict = {}
for i in range(2, 7):
    p12_y = label_transform(y, i)
    m = svm_train(p12_y, x, f'-s 0 -t 1 -c {C} -d {Q} -g 1 -r 1')
    p_label, p_acc, p_val = svm_predict(p12_y, x, m)
    ein = np.mean(np.array(p_label) != np.array(p12_y))
    ein_dict[i] = ein

ans = max(ein_dict, key=ein_dict.get)
print(ans)
```

```
nSV = 642, nBSV = 566
Total nSV = 642
Accuracy = 98.5143% (10344/10500) (classification)
....*...*||w||^2 = 146.432956

optimization finished, #iter = 7570
nu = 0.043973
obj = -388.502891, rho = 2.313424
nSV = 503, nBSV = 429
Total nSV = 503
Accuracy = 98.8762% (10382/10500) (classification)
5
```

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

def label_transform(y, target_label):
    return list(map(lambda x: 2*x-1, np.array(y) == target_label))

y, x = svm_read_problem('./letter.scale.tr')


C = 1
Q = 2
for i in range(2, 7):
    p12_y = label_transform(y, i)
    m = svm_train(p12_y, x, f'-s 0 -t 1 -c {C} -d {Q} -g 1 -r 1')
    p_label, p_acc, p_val = svm_predict(p12_y, x, m)
```

```
..*..*||w||^2 = 208.475556

optimization finished, #iter = 4921
nu = 0.052270
obj = -444.592560, rho = 0.396287
nSV = 588, nBSV = 511
Total nSV = 588
Accuracy = 98.8667% (10381/10500) (classification)
...*.*||w||^2 = 126.512266

optimization finished, #iter = 4145
nu = 0.031626
obj = -268.813999, rho = 3.726450
nSV = 368, nBSV = 298
Total nSV = 368
Accuracy = 99.3238% (10429/10500) (classification)
...*..*||w||^2 = 167.483630

optimization finished, #iter = 5745
nu = 0.043199
obj = -369.851393, rho = 0.978296
nSV = 499, nBSV = 420
Total nSV = 499
Accuracy = 99.0381% (10399/10500) (classification)
...*...*||w||^2 = 199.317761

optimization finished, #iter = 6221
nu = 0.057594
obj = -505.077850, rho = 1.142476
nSV = 642, nBSV = 566
Total nSV = 642
Accuracy = 98.5143% (10344/10500) (classification)
Total nSV = 499
Accuracy = 99.0381% (10399/10500) (classification)
...*...*||w||^2 = 199.317761

optimization finished, #iter = 6221
nu = 0.057594
obj = -505.077850, rho = 1.142476
nSV = 642, nBSV = 566
Total nSV = 642
Accuracy = 98.5143% (10344/10500) (classification)
....*...*||w||^2 = 146.432956

optimization finished, #iter = 7570
nu = 0.043973
obj = -388.502891, rho = 2.313424
nSV = 503, nBSV = 429
Total nSV = 503
Accuracy = 98.8762% (10382/10500) (classification)
```

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

def label_transform(y, target_label):
    return list(map(lambda x: 2*x-1, np.array(y) == target_label))


C = [0.01, 0.1, 1, 10, 100]
y, x = svm_read_problem('./letter.scale.tr')
p14_y = label_transform(y, 7)
y_test, x_test = svm_read_problem('./letter.scale.t')
p14_y_test = label_transform(y_test, 7)

eout_dict = {}
for c in C:
    m = svm_train(p14_y, x, f'-s 0 -t 2 -c {c} -g 1')
    p_label, p_acc, p_val = svm_predict(p14_y_test, x_test, m)
    eout_dict[c] = np.mean(np.array(p_label) != np.array(p14_y_test))

print(eout_dict)
ans = min(eout_dict, key=eout_dict.get)
print(ans)
```

```
nSV = 432, nBSV = 98
Total nSV = 432
Accuracy = 99.6% (4980/5000) (classification)
....*..*||w||^2 = 3387.578857

optimization finished, #iter = 6329
nu = 0.003247
obj = -1715.741274, rho = 4.525820
nSV = 336, nBSV = 1
Total nSV = 336
Accuracy = 99.46% (4973/5000) (classification)
{0.01: 0.0452, 0.1: 0.0452, 1: 0.0142, 10: 0.004, 100: 0.0054}
10
```

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

def label_transform(y, target_label):
    return list(map(lambda x: 2*x-1, np.array(y) == target_label))


gammma = [0.1, 1, 10, 100, 1000]
y, x = svm_read_problem('./letter.scale.tr')
p15_y = label_transform(y, 7)
y_test, x_test = svm_read_problem('./letter.scale.t')
p15_y_test = label_transform(y_test, 7)

eout_dict = {}
for g in gammma:
    m = svm_train(p15_y, x, f'-s 0 -t 2 -c 0.1 -g {g}')
    p_label, p_acc, p_val = svm_predict(p15_y_test, x_test, m)
    eout_dict[g] = np.mean(np.array(p_label) != np.array(p15_y_test))

print(eout_dict)
ans = min(eout_dict, key=eout_dict.get)
print(ans)
```

```
nSV = 10049, nBSV = 384
Total nSV = 10049
Accuracy = 95.48% (4774/5000) (classification)
......*.........*||w||^2 = 4.113193

optimization finished, #iter = 16680
nu = 0.073143
obj = -74.743403, rho = 0.996023
nSV = 10058, nBSV = 384
Total nSV = 10058
Accuracy = 95.48% (4774/5000) (classification)
{0.1: 0.0452, 1: 0.0452, 10: 0.0402, 100: 0.0452, 1000: 0.0452}
10
```

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

def label_transform(y, target_label):
    return list(map(lambda x: 2*x-1, np.array(y) == target_label))

def train_test_spilt(x, y):
    rng = np.random.default_rng()
    numbers = rng.choice(len(x), size=len(x), replace=False)
    x, y = np.array(x), np.array(y)
    # 200 tranining samples
    x_eval = x[numbers[:200]]
    y_eval = y[numbers[:200]]
    # rest evaluation samples
    x_train = x[numbers[200:]]
    y_train = y[numbers[200:]]

    return x_train, x_eval, y_train, y_eval


y, x = svm_read_problem('./letter.scale.tr')
x = np.array(x)
gammma = [0.1, 1, 10, 100, 1000]
p16_y = np.array(label_transform(y, 7))

ans = {
        0.1:0,
          1:0,
         10:0,
        100:0,
       1000:0
       }

for _ in range(500):
    x_train, x_eval, y_train, y_eval = train_test_spilt(x, p16_y)
    e_eval = {}
    for g in gammma:
        m = svm_train(y_train, x_train, f'-s 0 -t 2 -c 0.1 -g {g} -q')
        p_label, p_acc, p_val = svm_predict(y_eval, x_eval, m, '-q')
        e_eval[g] = 1 - p_acc[0] * 0.01
    print(e_eval)
    ans[min(e_eval, key=e_eval.get)] += 1

print(ans)
print(max(ans, key=ans.get))
```

{0.1: 0.020000000000000018, 1: 0.020000000000000018, 10: 0.020000000000000018, 100: 0.020000000000000018, 1000: 0.020000000000000018}
||w||^2 = 0.163447
||w||^2 = 10.001675
||w||^2 = 18.392545
||w||^2 = 4.237563
||w||^2 = 4.073052
{0.1: 0.020000000000000018, 1: 0.020000000000000018, 10: 0.020000000000000018, 100: 0.020000000000000018, 1000: 0.020000000000000018}
||w||^2 = 0.151654
||w||^2 = 9.594428
||w||^2 = 17.840398
||w||^2 = 4.181110
||w||^2 = 4.018682
{0.1: 0.04499999999999993, 1: 0.04499999999999993, 10: 0.040000000000000036, 100: 0.04499999999999993, 1000: 0.04499999999999993}
{0.1: 296, 1: 0, 10: 204, 100: 0, 1000: 0}
0.1

```python
import numpy as np
from dataclasses import dataclass

@dataclass
class AdaBoostStump():
    n_init: int

    def fit(self, x, y):
        self.x = x
        self.y = y
        self.alpha, self.little_g = self.run_adaboost_ds(self.x, self.y, self.n_init)

        self.little_g_errors = []
        # collect 0/1 error for little_g
        for g in self.little_g:
            self.little_g_errors += [self.error(x[:,g[1]], y, g[0], g[2])]
        self.min_g_error = min(self.little_g_errors)
        self.max_g_error = max(self.little_g_errors)

        # compute big G error
        self.big_G_error_in = self.ada_error(x, y, self.alpha, self.little_g)

        return self

    def run_adaboost_ds(self, x, y, n_init):
        weight = np.full((len(x),), 1/len(x))
        little_g = []
        alpha = []
        best_error = 1e15
        for _ in range(n_init):
            s, dim, theta, error = self.decision_stump(x, y, weight)
            # print(error)
            # renew weights
            little_g.append([s, dim, theta])
            scaling_factor = np.sqrt((1-error)/error)
            alpha.append(np.log(scaling_factor))

            if best_error > error:
                best_error = error

            # update weights
            for i in range(len(y)):
                if y[i] != s*self.sign(x[i][dim]-theta):
                    weight[i] *= scaling_factor
                else:
                    weight[i] /= scaling_factor

        return alpha, little_g

    def decision_stump(self, x, y, weight):
        weight_sum = np.sum(weight)
        # print(f'weight sum: {weight_sum}')

        dim_best_error = np.zeros(x.shape[1])
        dim_best_theta = np.zeros(x.shape[1])
        dim_best_s = np.zeros(x.shape[1])
        for dim in range(x.shape[1]):
            sort_idx = np.argsort(x[:,dim])
            x_sorted = x[sort_idx][:,dim]
            y_sorted = y[sort_idx]
            weight_sorted = weight[sort_idx]

            # calculate the theta = ninf condition
            # when theta is ninf, s = +1 every label equal to -1 is wrong, s = -1 label = +1 is wrong
            dp_pos = np.zeros(len(x))
            # init thetas
            thetas = [-1e15] + [(x_sorted[i]+x_sorted[i+1])/2 for i in range(x.shape[0]-1)]
            thetas = np.array(thetas)
            # print(thetas)
            cnt = 0
            for i, theta in enumerate(thetas):
                # init -inf error
                if i == 0:
                    dp_pos[i] = np.sum(weight_sorted[np.where(y_sorted == -1)])
                else:
                    if theta == x_sorted[i]:
                        dp_pos[i] = dp_pos[i-1]
                        cnt += 1
                    else:
                        dp_pos[i] = dp_pos[i-1] + np.sum([weight_sorted[i-1-j] * y_sorted[i-1-j] for j in
range(cnt+1)])
                        cnt = 0

            dp_neg = (weight_sum - dp_pos) / weight_sum
            dp_pos = dp_pos / weight_sum

            pos_min_idx = np.argmin(dp_pos)
            neg_min_idx = np.argmin(dp_neg)

            if dp_pos[pos_min_idx] <= dp_neg[neg_min_idx]:
                dim_best_s[dim] = 1
                dim_best_error[dim] = dp_pos[pos_min_idx]
                dim_best_theta[dim] = thetas[pos_min_idx]
            else:
                dim_best_s[dim] = -1
                dim_best_error[dim] = dp_neg[neg_min_idx]
                dim_best_theta[dim] = thetas[neg_min_idx]

        best_dim = np.argmin(dim_best_error)
        best_theta = dim_best_theta[best_dim]
        best_s = dim_best_s[best_dim]
        best_error = dim_best_error[best_dim]

        return best_s, best_dim, best_theta, best_error

    def bigG(self, x, alpha, little_g):
        temp = 0
        for idx, a in enumerate(alpha):
            s = little_g[idx][0]
            dim = little_g[idx][1]
            theta = little_g[idx][2]
            temp += a * s * self.sign(x[dim] - theta)

        return self.sign(temp)

    def ada_error(self, x, y, alpha, little_g):
        error = 0
        for idx, data in enumerate(x):
            if self.bigG(data, alpha, little_g) != y[idx]:
                error += 1

        return error/len(y)

    def error(self, x, y, s, theta):
        err = 0
        for idx, data in enumerate(x):
            if y[idx] != s * self.sign(data - theta):
                err += 1

        return  err/len(x)

    def calculate_eout(self, x, y):
        big_G_error_out = self.ada_error(x, y, self.alpha, self.little_g)
        return big_G_error_out

    def sign(self, value):
        if value >= 0:
            return 1
        else:
            return -1
```

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

In [ ]:
```
def label_transform(y, target_label):
    return list(map(lambda x: 2*x-1, np.array(y) == target_label))
```

In [ ]:
```
y, x = svm_read_problem('./letter.scale.tr')
y_test, x_test = svm_read_problem('./letter.scale.t')
y, x = np.array(y), np.array(x)
y_test, x_test = np.array(y_test), np.array(x_test)
```

In [ ]:
```
# find label 11 and label 26 index for training data
label_11_idx = np.array(np.where(y == 11)).flatten()
label_26_idx = np.array(np.where(y == 26)).flatten()
print(len(label_11_idx))
print(len(label_26_idx))
idx = np.concatenate((label_11_idx, label_26_idx))

x = x[idx]
y = y[idx]

temp = []
for i in range(len(x)):
    temp += [list(x[i].values())]

x = np.array(temp)
y = np.array(label_transform(y, 11))

# find label 11 and label 26 index for testing data
label_11_idx = np.array(np.where(y_test == 11)).flatten()
label_26_idx = np.array(np.where(y_test == 26)).flatten()
idx = np.concatenate((label_11_idx, label_26_idx))

x_test = x_test[idx]
y_test = y_test[idx]

temp = []
for i in range(len(x_test)):
    temp += [list(x_test[i].values())]

x_test = np.array(temp)
y_test = np.array(label_transform(y_test, 11))
```

```
404
378
```

In [ ]:
```
"""
Train AdaboostStump
"""
ada_model = AdaBoostStump(n_init=1000).fit(x, y)
```

In [ ]:
```
"""
P17
"""
```

```
ada_model.min_g_error
```

Out[ ]:
```
0.09846547314578005
```

In [ ]:
```
"""
P18
"""
ada_model.max_g_error
```

Out[ ]:
```
0.571611253196931
```

In [ ]:
```
"""
P19
"""
ada_model.big_G_error_in
```

Out[ ]:
```
0.0
```

In [ ]:
```
"""
P20
"""
eout = ada_model.calculate_eout(x_test, y_test)
eout
```

Out[ ]:
```
0.002793296089385475
```