HTML_2023_HW5

tags: `Personal`

```
Discuss with anonymous and TAs.
```

Support Vector Machines

P1

$$\min_{b,w,\xi} \ \frac{1}{2}w^Tw + C \cdot \sum_{n=1}^{N} \xi_n$$
$$\text{s.t.} \ \ y_n(w^Tz_n + b) \geq 1 - \xi_n \text{ and } \xi_n \geq 1 \ \forall n$$

Since we have the slack variable equations above, we know that ξ is used to record the number of margin violation, and it shows that

- For (x_n, y_n) voilate the margin: $\xi_n = 1 - y_n(w^Tz_n + b)$
- For (x_n, y_n) not voilate the margin: $\xi_n = 0$

To determine the number of misclassified examples, we need to consider the values of ξ_n, when $\xi_n > 1$, it indicates a misclassified example. Therefore, we can simply check the summation of misclassified examples by given $\xi = 1$, which means samples are on the boundaries, and the summation of ξ should not larger than n, which is the number of all the samples.

As a result, we have

$$
\begin{aligned}
&\sum_{n=1}^{N} \frac{\xi_n^*}{2} && \Rightarrow \sum_{n=1}^{N} \frac{1}{2} \neq n \\
&\sum_{n=1}^{N} \sqrt{\xi_n^*} && \Rightarrow \sum_{n=1}^{N} 1 = n \\
&\sum_{n=1}^{N} \xi_n^* && \Rightarrow \sum_{n=1}^{N} 1 = n \\
&\sum_{n=1}^{N} \lfloor \xi_n^* \rfloor && \Rightarrow \sum_{n=1}^{N} 1 = n \\
&\sum_{n=1}^{N} \log_2(1+\xi_n^*) && \Rightarrow \sum_{n=1}^{N} 1 = n
\end{aligned}
$$

As a result, only choice $\sum_{n=1}^{N} \frac{\xi_n^*}{2}$ is not the upper bound of the number of misclassified examples, and there are 4 correct outcomes reach the goal.

As a result, we should choose $[d]$ as our solution.

P2

Consider a soft-margin primal SVM, by complementary slackness, we have $\alpha_n(1-\xi_n-y_n(w^Tz_n+b))=0$ if $\forall n:\alpha_n^*=C\neq 0$, which entails that $1-\xi_n-y_n(w^Tz_n+b)=0$, hence, $y_nb=1-\xi_n-y_n\sum_{m=1}^{N}y_m\alpha_mK(x_n,x_m)$. Another restriction is that $\xi_n\geq 0, \xi_n=(1-y_n(w^Tz_n+b))\geq 0$, hence, $y_n(w^Tz_n+b)\leq 1$

Given $y_n=1$, the upper bound of b can be obtained, so we let $y_n=-1$

$$
y_nb = -1\cdot b = 1-\underbrace{\xi_n}_{\geq 0}+\sum_{m=1}^{N}y_m\alpha_mK(x_n,x_m)\geq 1+\sum_{m=1}^{N}y_m\alpha_mK(x_n,x_m)
$$

$$
b\geq -1-\sum_{m=1}^{N}y_m\alpha_m^*K(x_n,x_m)
$$

In order to find the smallest b^*, we have to take the maximum of the equation above, which means

$$
b^*\geq \max_{n:\,y_n<0}(-1-\sum_{m=1}^{N}y_m\alpha_m^*K(x_n^Tx_m))
$$

As a result, we should choose $[d]$ as our solution.

P3

Adding Lagrange multipliers α_n to P_2, we have

$$\mathcal{L}(\alpha, w, b) = \frac{1}{2}w^T w + C\sum_{n=1}^{N}\xi_n^2 + \sum_{n=1}^{N}\alpha_n(1 - \xi_n - y_n(w^T\phi(x_n) + b))$$
$$\frac{\partial \mathcal{L}(\alpha, w, b)}{\partial b} = -\sum_{n=1}^{N}\alpha_n y_n = 0$$
$$\frac{\partial \mathcal{L}(\alpha, w, b)}{\partial \xi_n} = 2C\xi_n - \alpha_n \Rightarrow \xi_n = \frac{\alpha_n}{2C}$$
$$\frac{\partial \mathcal{L}(\alpha, w, b)}{\partial w} = w - \sum_{n=1}^{N}\alpha_n y_n \phi(x_n) = 0 \Rightarrow w = \sum_{n=1}^{N}\alpha_n y_n \phi(x_n)$$

Since both ξ_n and α_n are the n-th element in the vector ξ and α_n, the optimal $\xi_n^* = \frac{\alpha_n^*}{2C}$ (vector).

As a result, we should choose $[a]$ as our solution.

P4

The $K_{d,s}(x, x^{'})$ actually means that how classifiers classify x and $x^{'}$ in the same group since classifiers classify x and $x^{'}$ into different groups. From the observation, if $x = x^{'}$, any θ will make both classified into the same group. For any L and R, $(L < R)$, there are $(R - L + 1)$ even integers between $2L$ and $2R$, and there are $(R - L)$ odd integers, which means there are $(R - L)$ candidate θ, and each θ can generate 2 decision stump. Therefore, $K_{ds}(x, x^{'}) = 2d(R - L)$ if $x = x^{'}$. Moreover, we can see that if a classifier $g_{s,i,\theta}$ is on the same side of x and $x^{'}$ (both x and $x^{'}$ are bigger or smaller than θ), this entails that a classifier $g_{s,i,\theta}$ classify x and $x^{'}$ different when it is between x and $x^{'}$, and there exists $\frac{|x_i - x_i^{'}|}{2}$ different choices of θ for fixed i and s. That means there are total $2\sum_{i=1}^{d}\frac{|x_i - x_i^{'}|}{2} = \|x - x^{'}\|_1$ classify x and $x^{'}$ different.

$$\begin{aligned}
K_{ds}(x, x^{'}) &= \left(\sum_{\text{All combinations}}[g_{s,i,\theta}(x) = g_{s,i,\theta}(x^{'}) =]\right) - \left(\sum_{\text{All combinations}}[g_{s,i,\theta}(x) \neq g_{s,i,\theta}(x^{'}) =]\right) \\
&= \underbrace{2d(R - L)}_{\text{All combinations}} - \|x - x^{'}\|_1 - \|x - x^{'}\|_1, \text{ where } x \neq x^{'} \\
&= 2d(R - L) - 2\|x - x^{'}\|_1
\end{aligned}$$

As a result, we should choose $[e]$ as our solution.

Bagging and Boosting

P5

In order to find the upper bound of $E_{out}(G)$, we have to classify a point incorrectly, and for $2M+1$ binary classifiers, there must be $\frac{(2M+1)+1}{2}$ classifiers that classified the points incorrectly. If there exists N points, there exists $(\sum_{t=1}^{2M+1} e_t)N$ points that classified incorrectly. Therefore, the number of points that classified incorrectly by aggregated binary classifiers can be represented as below.

$$\frac{(\sum_{t=1}^{2M+1} e_t)N}{\frac{(2M+1)+1}{2}}$$

And the upper bound of $E_{out}(G)$ equals to

$$\frac{(\sum_{t=1}^{2M+1} e_t)N}{\frac{(2M+1)+1}{2}N} = \frac{1}{M+1}\sum_{t=1}^{2M+1} e_t$$

As a result, we should choose $[b]$ as our solution.

P6

The probability P that every sample doesn't duplicated equals to

$$P = \frac{1127}{1127} \times \frac{1126}{1127} \times \frac{1125}{1127} \times \cdots \times \frac{1127-N}{1127} = \prod_{n=0}^{N-1}$$

And the probability of samples that duplicate at least once equals to $1-P$, therefore, we have

$$\frac{\binom{1127}{n} \cdot n!}{1127^n} \leq (1-0.75) = 0.25$$

```python
from math import comb, factorial

N=1127;P=0.75;target_n = 0
for n in range(1,100):
    if (((factorial(n)*comb(N, n))/(N**n))<=(1-P)):
        print("Probability: ", (factorial(n)*comb(N, n))/(N**n))
        target_n = n; break

print("Smallest N:", target_n)
```

```
Probability:  0.24920995191245734
Smallest N: 56
```

As a result, we should choose $[b]$ as our solution.

P7

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

Since $u_n \geq 0$, we can rewrite the equation as below.

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

Then we can let $(\tilde{x}_n, \tilde{y}_n) = (\sqrt{u_n}\tilde{x}_n, \sqrt{u_n}\tilde{y}_n)$, then the equation above can be represented as below.

$$E_{in}^u(w) = \frac{1}{N}\sum_{n=1}^{N}(\tilde{y}_n - w^T\tilde{x}_n)^2$$

As a result, we should choose $[c]$ as our solution.

P8

[a]

- Gini index for the first part: $1 - (1)^2 - (0)^2 = 0$
- Gini index for the second part: $1 - (0.5)^2 - (0.5)^2 = 0.5$
- Overall Gini index: $0 \cdot 0.5 + 0.5 \cdot 0.5 = 0.25$

[b]

- Gini index for the first part: $1-(0.8)^2-(0.2)^2=0.32$
- Gini index for the second part: $1-(0.75)^2-(0.25)^2=0.375$
- Overall Gini index: $0.32\cdot 0.7+0.375\cdot 0.3=0.3365$

[c]

- Gini index for the first part: $1-(0.7)^2-(0.3)^2=0.42$
- Gini index for the second part: $1-(0)^2-(1)^2=0$
- Overall Gini index: $0.42\cdot 0.9+0\cdot 0.1=0.378$

[d]

- Gini index for the first part: $1-(0.8)^2-(0.2)^2=0.32$
- Gini index for the second part: $1-(0.9)^2-(0.1)^2=0.18$
- Overall Gini index: $0.32\cdot 0.8+0.18\cdot 0.2=0.292$

[e]

- Gini index for the first part: $1-(0.9)^2-(0.1)^2=0.18$
- Gini index for the second part: $1-(0.9)^2-(0.1)^2=0.18$
- Overall Gini index: $0.18\cdot 0.8+0.18\cdot 0.2=0.18$

Since `[e]` has minimum Gini index, `[e]` has the best branch for building a CART decision tree.

As a result, we should choose $[e]$ as our solution.
Ref: Day 22 : 決策樹 (https://ithelp.ithome.com.tw/articles/10276079)

P9

Assume that when iteration at t, there are a incorrect, and a terms correct, and from the lecture slide `Lecture 12: Bagging and Boosting P.34`, we have

$$
\begin{aligned}
\blacklozenge_t &= \sqrt{\frac{1-\epsilon_t}{\epsilon_t}} \geq 1, 0 \leq \epsilon_t \leq \frac{1}{2} \\
U_{t+1} &= \sum_{n=1}^{N} u_n^{t+1} \\
&= \sum_{n=1}^{N} u_n^t \times \blacklozenge_t \cdot [\![y_n \neq g_t(x_n)]\!] + \sum_{n=1}^{N} u_n^t \times \frac{1}{\blacklozenge_t} \times [\![y_n \neq g_t(x_n)]\!] \\
&= \epsilon_t \times \blacklozenge_t \times \sum_{n=1}^{N} u_n^t + \frac{1-\epsilon_t}{\blacklozenge_t} \times \sum_{n=1}^{N} u_n^t \\
&= U_t \times (\epsilon_t \blacklozenge_t + \frac{1-\epsilon_t}{\blacklozenge_t}) \\
&= 2U_t \sqrt{\epsilon_t(1-\epsilon_t)} \\
&= 2\sqrt{\epsilon_t(1-\epsilon_t)}(2U_{t-1}\sqrt{\epsilon_{t-1}(1-\epsilon_{t-1})}) \\
&= 2^T \prod_{t=1}^{T} \sqrt{\epsilon_t(1-\epsilon_t)}
\end{aligned}
$$

As a result, we should choose $[d]$ as our solution.

P10

From the lecture slide, we have

$$
\min_{\eta} \frac{1}{N} \sum_{n=1}^{N} \operatorname{err}(\sum_{\tau=1}^{t-1} \alpha_\tau g_\tau(x_n) + \eta g_t(x_n), y_n) \text{ with } \operatorname{err}(s, y) = (s-y)^2
$$

and we can rewrite the equation as below.

$$
\begin{aligned}
\text{eq1} \Rightarrow \min_{\eta} \frac{1}{N} \sum_{n=1}^{N} (s_n + \eta g_t(x_n) - y_n)^2 \\
\frac{\partial(\text{eq1})}{\partial \eta} &= \frac{1}{N} \sum_{n=1}^{N} \cdot 2 \cdot (s_n + \eta g_t(x) - y_n) \cdot (g_t(x)) = 0 \\
&\Rightarrow \sum_{n=1}^{N} (s_n + \eta g_t(x) - y_n) \cdot (g_t(x)) = 0 \\
\because s_{n+1} &= s_n + \eta g_t(x) \\
&\therefore \Rightarrow \sum_{n=1}^{N} (s_{n+1} - y_n) \cdot g_t(x) = 0
\end{aligned}
$$

As a result, we should choose $[b]$ as our solution.

Experiments with Soft-Margin SVM and AdaBoos

P11

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

def main(args):
    y_train, x_train = svm_read_problem(args['svm_train'])
    target = 1

    for i in range(len(y_train)):
        y_train[i] = 1 if y_train[i] == target else 0

    prob = svm_problem(y_train, x_train)
    param = svm_parameter('-s 0 -t 0 -c {} -q'.format(args['C']))
    m = svm_train(prob, param)
    support_vector_coefficients = m.get_sv_coef()
    support_vectors = m.get_SV()

    height = len(support_vector_coefficients)
    width = 0
    primal = list()

    for i in range(len(support_vectors)):
        width = max(max(support_vectors[i].keys()), width)

    for i in range(width):
        accuracy = 0
        for j in range(height):
            if(i+1) in support_vectors[i]:
                accuracy += support_vectors[j][i+1] * support_vector_coefficients[j
        primal.append(accuracy)
    primal = np.array(primal)

    print("||w||: ", sum(primal * primal) ** 0.5)
```

```
if __name__ == '__main__':
    args = {
    'C': 1,
    'svm_train': "svm_letter.scale.tr",
    'svm_test': "svm_letter.scale.t",
    }

    main(args)
```

```
||w||:  6.305469435447087
```

As a result, we should choose $[c]$ as our solution.

P12

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

def main(args):
    for classifier in range(2,7):
        print(f'=== "{classifier}" versus "not {classifier}"')
        y_train, x_train = svm_read_problem(args['svm_train'])
        for type in range(len(y_train)):
            y_train[type] = 1 if y_train[type] == classifier else 0

        prob = svm_problem(y_train, x_train)
        param = svm_parameter('-s 0 -t 1 -d {Q} -g 1 -r 1 -c {C} -q'.format(C=args[
        m = svm_train(prob, param)
        p_label, p_acc, p_val = svm_predict(y_train, x_train, m)
        print("sVM: ", len(m.get_SV()))
```

```
if __name__ == '__main__':
    args = {
    'adaboost_train': "adaboost_letter.scale.tr",
    'adaboost_test': "adaboost_letter.scale.t",
    'C': 1,
    'Q': 2,
    'svm_train': "svm_letter.scale.tr",
    'svm_test': "svm_letter.scale.t",
    }

    main(args)
```

```
=== "2" versus "not 2"
Accuracy = 98.8667% (10381/10500) (classification)
sVM:  589
=== "3" versus "not 3"
Accuracy = 99.3238% (10429/10500) (classification)
sVM:  368
=== "4" versus "not 4"
Accuracy = 99.0381% (10399/10500) (classification)
sVM:  497
=== "5" versus "not 5"
Accuracy = 98.5143% (10344/10500) (classification)
sVM:  643
=== "6" versus "not 6"
Accuracy = 98.8762% (10382/10500) (classification)
sVM:  502
```

As a result, we should choose $[d]$ as our solution.

P13

We use the same code above.

```
=== "2" versus "not 2"
Accuracy = 98.8667% (10381/10500) (classification)
sVM:  589
=== "3" versus "not 3"
Accuracy = 99.3238% (10429/10500) (classification)
sVM:  368
=== "4" versus "not 4"
Accuracy = 99.0381% (10399/10500) (classification)
sVM:  497
=== "5" versus "not 5"
Accuracy = 98.5143% (10344/10500) (classification)
sVM:  643
=== "6" versus "not 6"
Accuracy = 98.8762% (10382/10500) (classification)
sVM:  502
```

As a result, we should choose $[b]$ as our solution.

P14

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

def main(args):
    choices = [0.01, 0.1, 1, 10, 100]
    y_train, x_train = svm_read_problem(args['svm_train'])
    y_test, x_test = svm_read_problem(args['svm_test'])

    for type in range(len(y_train)):
        y_train[type] = 1 if y_train[type] == args['classifier'] else 0
    for type in range(len(y_test)):
        y_test[type] = 1 if y_test[type] == args['classifier'] else 0

    for choice in choices:
        print(f'=== choice: {choice}')
        prob = svm_problem(y_train, x_train)
        param = svm_parameter('-s 0 -t 2 -g {gamma} -c {C} -q'.format(C=choice, gam
        m = svm_train(prob, param)
        p_label, p_acc, p_val = svm_predict(y_test, x_test, m)
```

```
if __name__ == '__main__':
    args = {
    'classifier': 7,
    'gamma': 1,
    'svm_train': "svm_letter.scale.tr",
    'svm_test': "svm_letter.scale.t",
    }

    main(args)
```

```
=== choice: 0.01
Accuracy = 95.48% (4774/5000) (classification)
=== choice: 0.1
Accuracy = 95.48% (4774/5000) (classification)
=== choice: 1
Accuracy = 98.58% (4929/5000) (classification)
=== choice: 10
Accuracy = 99.6% (4980/5000) (classification)
=== choice: 100
Accuracy = 99.46% (4973/5000) (classification)
```

As a result, we should choose $[d]$ as our solution.

P15

We use the same code above, then update the `choices`, `param`, and the parameter of args. The updated lines show below.

```
choices = [0.1, 1, 10, 100, 1000]
param = svm_parameter('-s 0 -t 2 -g {gamma} -c {C} -q'
                      .format(C=args['C'], gamma=choice))

if __name__ == '__main__':
    args = {
    'C': 0.1,
    'classifier': 7,
    'svm_train': "svm_letter.scale.tr",
    'svm_test': "svm_letter.scale.t",
    }

    main(args)
```

```
=== choice: 0.1
Accuracy = 95.48% (4774/5000) (classification)
=== choice: 1
Accuracy = 95.48% (4774/5000) (classification)
=== choice: 10
Accuracy = 95.98% (4799/5000) (classification)
=== choice: 100
Accuracy = 95.48% (4774/5000) (classification)
=== choice: 1000
Accuracy = 95.48% (4774/5000) (classification)
```

As a result, we should choose $[c]$ as our solution.

P16

```
import numpy as np
import random
from libsvm.svmutil import *
from tqdm import trange

def main(args):
    choices = [0.1, 1, 10, 100, 1000]
    y_train, x_train = svm_read_problem(args['svm_train'])

    for type in range(len(y_train)):
        y_train[type] = 1 if y_train[type] == args['classifier'] else 0

    gamma_count = [0, 0, 0, 0, 0]
    for i in trange(args['repeat_time']):
        sampling = random.sample(range(len(y_train)), args['sampling'])
        training_id = [i for i in range((len(y_train))) if i not in sampling]
        validation_id = [i for i in range((len(y_train))) if i in sampling]

        min = 1
        min_gamma = 0
        for i, g in enumerate(choices):
            prob = svm_problem([y_train[idx] for idx in training_id], [x_train[idx]
            param = svm_parameter('-s 0 -t 2 -g {gamma} -c {C} -q'.format(C=args['C
            m = svm_train(prob, param)
            p_label, p_acc, p_val = svm_predict([y_train[idx] for idx in validatior
            print(p_acc)
            if p_acc[1] < min:
                min_gamma = i
                min = p_acc[1]
        gamma_count[min_gamma] += 1
        print("gamma_count:", gamma_count)
    print("final gamma_count:", gamma_count)
```

```
if __name__ == '__main__':
    args = {
    'C': 0.1,
    'classifier': 7,
    'sampling': 200,
    'svm_train': "svm_letter.scale.tr",
    'svm_test': "svm_letter.scale.t",
    'repeat_time': 500,
    }

    main(args)
```

```
final gamma_count: [305, 0, 195, 0, 0]
```

As a result, we should choose $[a]$ as our solution.

P17

```
import math
import numpy as np
from libsvm.svmutil import *
from tqdm import trange

def adaboost(args, x, y):
    u = np.ones((args['size'], )) / args['size']   # weights
    E_in_u, alpha = np.zeros((args['T'],)), np.zeros((args['T'],))
    s, i, theta = np.zeros((args['T'],)), np.zeros((args['T'],), dtype=int), np.zer
    for t in trange(args['T']):
        # get optimal parameters
        E_in_u[t], s[t], i[t], theta[t] = decision_stump_multi(args, x, y, u)
        diamond = np.sqrt((1-E_in_u[t])/E_in_u[t])
        # update u, which are weights
        for idx in range(len(y)):
            if y[idx] != s[t] * sign(x[idx][i[t]] - theta[t]):
                u[idx] *= diamond
            else:
                u[idx] /= diamond
        alpha[t] = np.log(diamond)
    return E_in_u, s, i, theta, alpha
```

```python
def decision_stump(args, x, y, u):
    # h(x) = s * sign (x - theta)
    s = 1; theta = float('-inf'); error = float('inf')
    sorted_x = np.array(sorted(x))
    theta_list = np.array([float('-inf')] + [((sorted_x[i]+sorted_x[i+1])/2) for i

    for theta_hypothesis in theta_list:
        y_pos = np.where(x >= theta_hypothesis, 1, -1)
        y_neg = np.where(x < theta_hypothesis, 1, -1)
        error_pos = np.sum(u[np.where(y_pos != y)])/np.sum(u)
        error_neg = np.sum(u[np.where(y_neg != y)])/np.sum(u)

        if error_pos > error_neg:
            if error_neg < error:
                error = error_neg
                theta = theta_hypothesis
                s = -1
        else:
            if error_pos < error:
                error = error_pos
                theta = theta_hypothesis
                s = 1
    return error, s, theta

def decision_stump_multi(args, x, y, u):
    s = np.zeros((args['dimension'],))
    theta = np.zeros((args['dimension'],))
    error = np.zeros((args['dimension'],))

    for dim in range(args['dimension']):
        error[dim], s[dim], theta[dim] = decision_stump(args, x[:,dim], y, u)
    i = np.argmin(error)
    return error[i], s[i], i, theta[i]
```

```
def get_E_in_list(args, x, y, s, i, theta):
    error_list = list()
    for t in range(args['T']):
        error = 0
        for n in range(len(y)):
            if(y[n] != (s[t] * np.sign(x[n][i[t]] - theta[t]))):
                error += 1
        error_list.append(error/len(y))
    return error_list

def predict_G(args, x, y, s, i, theta, alpha):
    error = 0
    for n in range(len(x)):
        temp = 0
        for alpha_n in range(len(alpha)):
            temp += alpha[alpha_n] * (s[alpha_n] * np.sign(x[n][i[alpha_n]] - theta
        guess = np.sign(temp)
        if(y[n] != guess):
            error += 1
    return error/len(y)

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

```
def transformation(args, y, x):
    transformed_x, transformed_y = list(), list()
    # idx:value => value only
    for n in range(len(y)):
        if(y[n] == args['label_pos'] or y[n] == args['label_neg']):
            transformed_y.append(y[n])
            row = list()
            for index, value in x[n].items():
                row.append(value)
            transformed_x.append(row)
    # one-versus-one: label "label_pos" versus label "label_neg"
    for label in range(len(transformed_y)):
        transformed_y[label] = 1 if transformed_y[label] == args['label_pos'] else
    return np.array(transformed_y), np.array(transformed_x)

def main(args):
    y_train, x_train = transformation(args, *svm_read_problem(args['adaboost_train'
    y_test, x_test = transformation(args, *svm_read_problem(args['adaboost_test']))

    # hyper-parameters
    args['dimension'] = x_train.shape[1]
    args['size'] = len(x_train)
    # adaboost and decision_stump
    E_in_u, s, i, theta, alpha = adaboost(args, x_train, y_train)   # only return c
    # P17, P18
    E_in_list = get_E_in_list(args, x_train, y_train, s, i, theta)
    print("min E_in(g_t): ", min(E_in_list))
    print("max E_in(g_t): ", max(E_in_list))
    # P19, P20
    E_in_G = predict_G(args, x_train, y_train, s, i, theta, alpha)
    E_out_G = predict_G(args, x_test, y_test, s, i, theta, alpha)
    print("E_in(G): ", E_in_G)
    print("E_out(G): ", E_out_G)
    print("===Fuck you HTML===")
```

```python
if __name__ == '__main__':
    args = {
    'adaboost_train': "adaboost_letter.scale.tr",
    'adaboost_test': "adaboost_letter.scale.t",
    'dimension': 16,
    'label_pos': 11,
    'label_neg': 26,
    'size': 0,
    'T': 1000,
    'type': 1,  # 1: best, 0: worst
    }

    main(args)
```

```
min E_in(g_t):  0.09846547314578005
max E_in(g_t):  0.571611253196931
E_in(G):  0.0
E_out(G):  0.002793296089385475
```

As a result, we should choose $[a]$ as our solution.

P18

We use the same code above.

```
min E_in(g_t):  0.09846547314578005
max E_in(g_t):  0.571611253196931
E_in(G):  0.0
E_out(G):  0.002793296089385475
```

As a result, we should choose $[c]$ as our solution.

P19

We use the same code above.

```
min E_in(g_t):  0.09846547314578005
max E_in(g_t):  0.571611253196931
E_in(G):  0.0
E_out(G):  0.002793296089385475
```

As a result, we should choose $[a]$ as our solution.

P20

We use the same code above.

```
min E_in(g_t):  0.09846547314578005
max E_in(g_t):  0.571611253196931
E_in(G):  0.0
E_out(G):  0.002793296089385475
```

As a result, we should choose $[a]$ as our solution.