HTML Homework 1

B10705005 資工二 陳思如

Discuss with: 陳妍姍、陳竹欣

1.

Ans: (a)
(a) pattern: requests; definition: users' words may differ, not easily programmable; data: the question and the related response
(b) no pattern
(c) programmable question
(d) programmable question

2.

Ans: (d)

if $\frac{-y_n(t)w_t^T x_n(t)}{|x_n(t)|^2} \in \mathbb{Z}$,

then $\frac{-y_n(t)w_t^T x_n(t)}{|x_n(t)|^2} = \lceil \frac{-y_n(t)w_t^T x_n(t)}{|x_n(t)|^2} \rceil < \lfloor \frac{-y_n(t)w_t^T x_n(t)}{|x_n(t)|^2} + 1 \rfloor$

if $\frac{-y_n(t)w_t^T x_n(t)}{|x_n(t)|^2} \notin \mathbb{Z}$,

then $\frac{-y_n(t)w_t^T x_n(t)}{|x_n(t)|^2} < \lceil \frac{-y_n(t)w_t^T x_n(t)}{|x_n(t)|^2} \rceil = \lfloor \frac{-y_n(t)w_t^T x_n(t)}{|x_n(t)|^2} + 1 \rfloor$

then we could know $\frac{-y_n(t)w_t^T x_n(t)}{|x_n(t)|^2} < \lfloor \frac{-y_n(t)w_t^T x_n(t)}{|x_n(t)|^2} + 1 \rfloor$

Also, we can ensure that w_{t+1}^T is correct if $y_n(t)w_{t+1}^T x_n(t) > 0$, which means $w_{t+1}^T x_n(t)$ has the same sign with $y_n(t)$

if we take $w_{t+1}^T = w_t + y_n(t)x_n(t)(\frac{-y_n(t)w_t^T x_n(t)}{|x_n(t)|^2})$,

then $y_n(t)w_{t+1}^T x_n(t) = y_n(t)w_t^T x_n(t) - y_n(t)w_t^T x_n(t) = 0$,

which means that we need to let w_{t+1}^T become bigger so that the $w_{t+1}^T x_n(t)y_n(t)$ will be greater than 0.

In this case, we have to choose $w_{t+1} = w_t + y_n(t)x_n(t) \cdot \lfloor \frac{-y_n(t)w_t^T x_n(t)}{|x_n(t)|^2} + 1 \rfloor$ to satisfy our condition.

3.

Ans: (c)

$T \leq (\frac{R}{\rho})^2$

$R = \max_n |x_n|$

$\rho = \min_n y_n w_f^T x_n$

by the description,

$\rho_z = \min_n \frac{y_n w_f^T z_n}{|w_f|}$

$R = \max_n |z_n|^2 = 1$

$\because z_n = \frac{x_n}{|x_n|}$

$\therefore |z_n| = (\frac{x_n}{|x_n|})^2 = 1$

then,

$T <= (\frac{R}{\rho})^2 = (\frac{1}{\rho_z})^2 = \frac{1}{\rho_z^2}$

4.

Ans: (b)

$U_{orig} = (\frac{R}{\rho})^2 = (\frac{\max_n |x_n|}{\min_n y_n w_f^T x_n})^2 = (\frac{\max_n |x_n|}{\min_n y_n w_f^T \frac{x_n}{|x_n|} |x_n|})^2$

$U = (\frac{1}{\rho_z})^2 = (\frac{1}{\min_n \frac{y_n w_f^T z_n}{|w_f|}})^2 = (\frac{1}{\min_n \frac{y_n w_f^T x_n}{|w_f||x_n|}})^2$

if $y_n w_f^T x_n > 0$,

then $U_{orig} \geq (\frac{\max_n |x_n|}{\min_n y_n w_f^T \frac{x_n}{|x_n|}(\min_n |x_n|)})^2 = (\frac{\max_n |x_n|}{\rho_z(\min_n |x_n|)})^2 \geq (\frac{1}{\rho_z})^2 = U$

if $y_n w_f^T x_n < 0$,

then $U_{orig} \geq (\frac{\max_n |x_n|}{\min_n y_n w_f^T \frac{x_n}{|x_n|}(\max_n |x_n|)})^2 = (\frac{\max_n |x_n|}{\rho_z(\max_n |x_n|)})^2 = (\frac{1}{\rho_z})^2 = U$

which could tell from all the two cases,

that $U_{orig} \geq U$

5.

Ans: (c)
For PLA in training examples,

w	x	y	$w_f x$	$sign(w_f x)$	w_{t+1}
(0,0,0)	(1,-2,2)	-1	0	1	(-1,2,-2)
(-1,2,-2)	(1,-1,2)	-1	-7	-1	(-1,2,-2)
(-1,2,-2)	(1,2,0)	1	3	1	(-1,2,-2)
(-1,2,-2)	(1,-1,0)	-1	-3	-1	(-1,2,-2)
(-1,2,-2)	(1,1,1)	1	-1	-1	(0,3,-1)

$$w_{PLA} = (0, 3, -1)$$

For PAM in training examples,

w	x	y	$w_f x$	$y_n w_f x$	γ	w_{t+1}
(0,0,0)	(1,-2,2)	-1	0	0	5	(-1,2,-2)
(-1,2,-2)	(1,-1,2)	-1	-7	7	5	(-1,2,-2)
(-1,2,-2)	(1,2,0)	1	3	3	5	(0,4,-2)
(0,4,-2)	(1,-1,0)	-1	-4	4	5	(-1,5,-2)
(-1,5,-2)	(1,1,1)	1	2	2	5	(0,6,-1)

$$w_{PAM} = (0, 6, -1)$$

For testing examples

x	y	w_{PLA} **result**	w_{PAM} **result**
$(1, \frac{1}{2}, 2)$	1	X	O
$(1, \frac{1}{4}, 1)$	1	X	O
$(1, \frac{1}{2}, 0)$	1	O	O
$(1, -\frac{1}{2}, 1)$	-1	O	O

$\therefore$ two test examples are wrongly predicted by PLA but correctly predicted by PAM

6.

Ans: (a)
∵ rating is [1,5], so the rating are continuous real numbers
∴ we should use regression

∵ with given existing rating from some viewers
∴ it is supervised learning

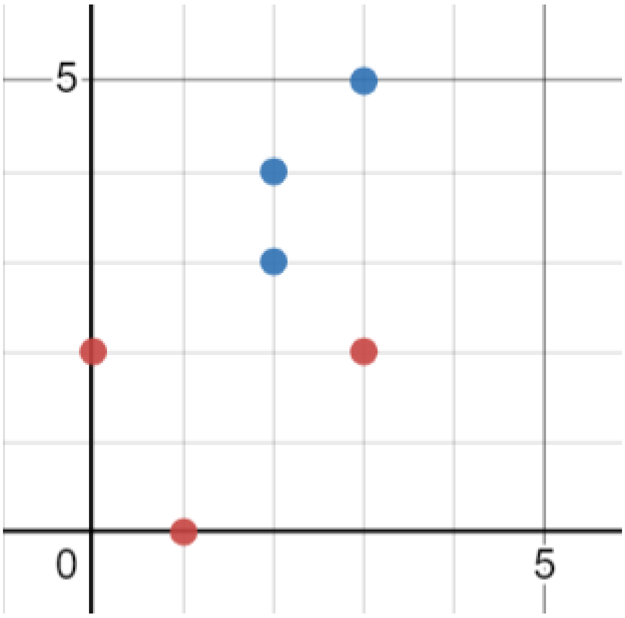
∴ it should be supervised regression learning problem

7.

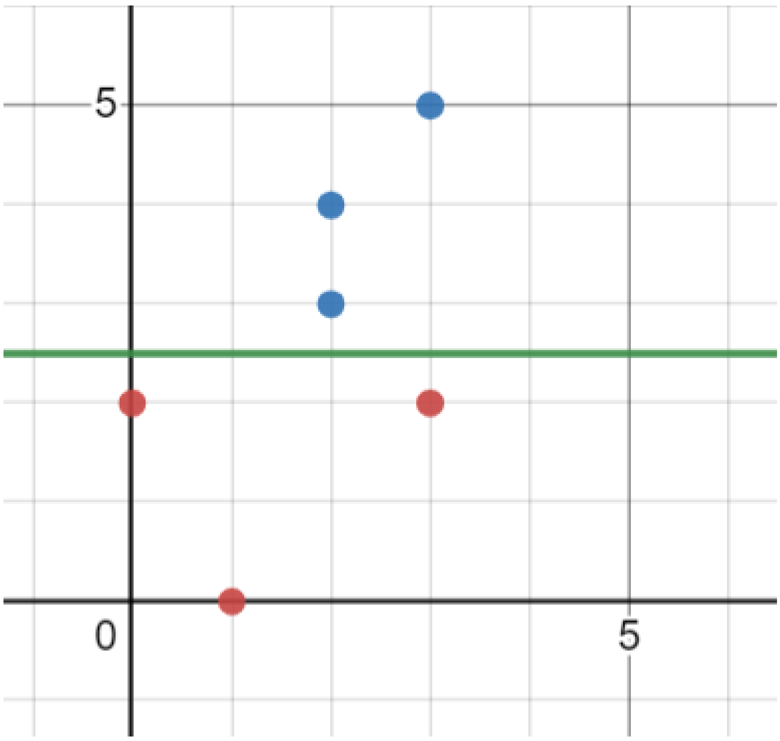
Ans: (b)
∵ the labeler has to always decide two of the outputs which are better
∴ the label will only have two category and one belongs to good and the other belongs to bad

8.

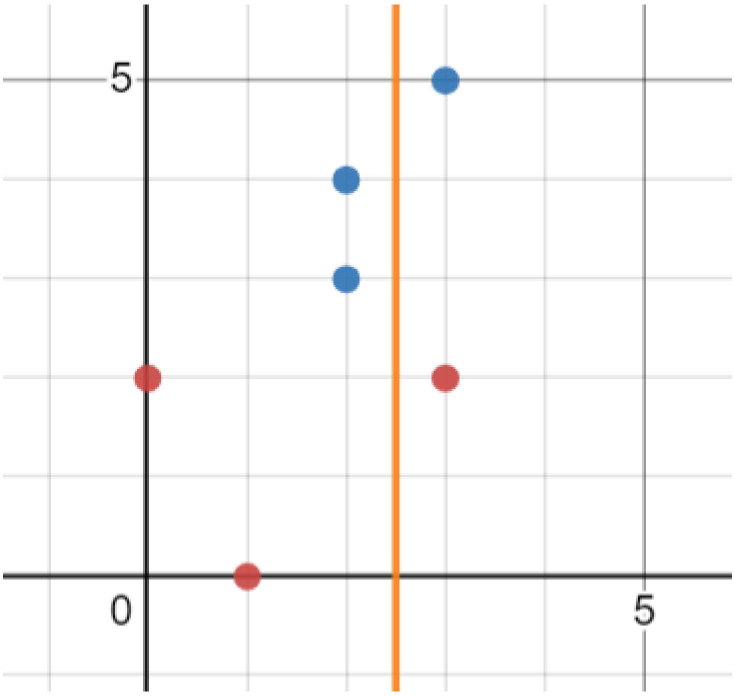
Ans: (e)



the above image is the universe example μ, red for $y=1$, blue for $y=-1$



if we draw a perceptron hypothesis at $y=2.5$,
then we can make $E_{in}(g)=0$ with any three examples from μ
and $E_{ots}(g)=\frac{0}{3}=0$ with the other three examples.

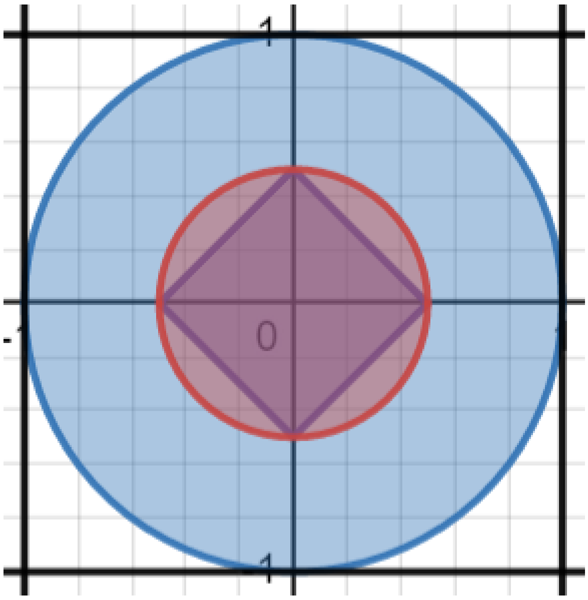


if we draw a perceptron hypothesis at $x = 2.5$,
then we can make $E_{in}(g) = 0$ with x_2, x_4, and x_5 from μ
and $E_{ots}(g) = \frac{3}{3} = 0$ with the other three examples.

$\therefore$ the smallest and largest $E_{ots}(g) = (0, \frac{1}{3})$

9.

Ans: (d)



the above image is for the target function region,

black lines are the boundaries, red is $f(x)$, blue is $h_1(x)$, purple is $h_2(x)$.

$$E_{out}(h_1) = \frac{region(h_1(x) - f(x))}{all\ region} = \frac{\pi - \frac{1}{4}\pi}{4} = \frac{\frac{3}{4}pi}{4} = \frac{3}{16}\pi$$

$$E_{out}(h_2) = \frac{region(f(x) - h_2(x))}{all\ region} = \frac{\frac{1}{4}\pi - \frac{1}{2}}{4} = \frac{\frac{pi-2}{4}}{4} = \frac{\pi - 2}{16}$$

10.

Ans: (b)

to let $E_{in}(h_1) = E_{in}(h_2) = 0$, we have to choose the points region that h_1 and h_2 have the same sign which is the region of all white and the h_2

Probability of choosing one point = $\frac{same\ sign\ region}{all\ region} = \frac{(4-\pi)+\frac{1}{2}}{4} = \frac{4.5-\pi}{4}$

$\therefore$ the probability of choosing four examples = $\frac{4.5-\pi}{4}^4 = 0.0133 \approx 0.01$

11.

Ans: (d)

by Hoeffding's Inequality,

$$\mathbb{P}[|E_{in} - E_{out}| > \epsilon] \leq 2\exp(-2\epsilon^2 N)$$

and we know that $E_{in} = \frac{darts\ in\ the\ circle}{all\ darts}$ and $E_{out} = \frac{\pi}{4}$ and $\epsilon = \frac{10^{-2}}{4} = 0.0025$ and $P[|E_{in} - E_{out}| > \epsilon] < 1 - 0.99 = 0.01$

then we solve the equation:

$$2\exp(-2(0.0025)^2 N) < 0.01$$

we then get $N > 423864$

12.

Ans: (d)

by Multiple-bin Hoeffding's Inequality,

$P[|E_{in} - E_{out}| > \epsilon] \leq 2M\exp(-2\epsilon^2 N)$

and we know that $P[|E_{in} - E_{out}| > \epsilon] = 1 - (1-\delta) = \delta$,

also that the p_m^* is the largest reward probability, so that the $|E_{in} - E_{out}|$ will only be suitable with only one side, that's because $p_m^* >= p_m$. Then this will make our estimation error $= \frac{\epsilon}{2}$.

then we solve the equation:

$$2M\exp(-2(\frac{\epsilon}{2})^2 N) \geq \delta$$

we then get $N \leq \frac{2}{\epsilon^2}\ln\frac{2M}{\delta}$

13.

Ans: (b)

source code:

```
#import libraries:
import numpy
from numpy import loadtxt
import random
from random import randint
import statistics
import math

#define the function to calculate w_f x_n:
def cal(w, x):
  sum = 0
  sum = float(sum)
  for i in range(11):
    sum += w[i]*x[i]
  return sum

#some initialization for N and x_n:
#N
N = 256
#xn
file = open('hw1_train.txt', 'r')
data = []
for line in file:
  number_strings = line.split()
  numbers = [float(n) for n in number_strings]
  data.append(numbers)

#we have x_0 = 1 and M = N/2
#initialize x_0 and add it to every x_n:
x0 = 1
for row in data:
    row.insert(0, x0)
data_n = numpy.array(data)

#run the experiment with the conditions and get the average E_{in}(w_{PLA}):
sumEin = float(0)
for E in range(1000):
    #w0 initialization
    w = [0]*11
    #M iteration
    M = int(N/2)
    cnt = 0
    while True:
        #random pick a xn
        x = randint(0, 255)
        #calculate the wfxn
        num = cal(w, data_n[x])
        #check sign
        if (num >= 0 and data_n[x][11] < 0) or (num < 0 and data_n[x][11] > 0):
            for j in range(11):
                w[j] = w[j] + data_n[x][j]*data_n[x][11]
```

```
            cnt = 0
        else:
            cnt += 1
        if cnt >= M: #stop at M correct consecutive check
            break

    #calculate Ein
    sum = float(0)
    for i in range(N):
        num = cal(w, data_n[i])
        if (num >= 0 and data_n[i][11] < 0) or (num < 0 and data_n[i][11] > 0):
            sum += 1
    sum = float(sum)
    sum = float(sum/N)
    sumEin += sum

#calculate average Ein
print(sumEin/1000)
```

I get average $E_{in}(w_{PLA}) = 0.01978125 \approx 0.02$

14.

Ans: (a)

source code:

```
#import libraries:
import numpy
from numpy import loadtxt
import random
from random import randint
import statistics
import math

#define the function to calculate w_f x_n:
def cal(w, x):
  sum = 0
  sum = float(sum)
  for i in range(11):
    sum += w[i]*x[i]
  return sum

#some initialization for N and x_n:
#N
N = 256
#xn
file = open('hw1_train.txt', 'r')
data = []
for line in file:
  number_strings = line.split()
  numbers = [float(n) for n in number_strings]
  data.append(numbers)

#we have x_0 = 1 and M = 4N
#initialize x_0 and add it to every x_n:
x0 = 1
for row in data:
    row.insert(0, x0)
data_n = numpy.array(data)

#run the experiment with the conditions and get the average E_{in}(w_{PLA}):
sumEin = float(0)
for E in range(1000):
    #w0 initialization
    w = [0]*11
    #M iteration
    M = int(4*N)
    cnt = 0
    while True:
        #random pick a xn
        x = randint(0, 255)
        #calculate the wfxn
        num = cal(w, data_n[x])
        #check sign
        if (num >= 0 and data_n[x][11] < 0) or (num < 0 and data_n[x][11] > 0):
            for j in range(11):
                w[j] = w[j] + data_n[x][j]*data_n[x][11]
```

```
            cnt = 0
        else:
            cnt += 1
        if cnt >= M: #stop at M correct consecutive check
            break

    #calculate Ein
    sum = float(0)
    for i in range(N):
        num = cal(w, data_n[i])
        if (num >= 0 and data_n[i][11] < 0) or (num < 0 and data_n[i][11] > 0):
            sum += 1
    sum = float(sum)
    sum = float(sum/N)
    sumEin += sum

#calculate average Ein
print(sumEin/1000)
```

I get average $E_{in}(w_{PLA}) = 0.000234375 \approx 0.00020$

15.

Ans: (d)

source code:

```
#import libraries:
import numpy
from numpy import loadtxt
import random
from random import randint
import statistics
import math

#define the function to calculate w_f x_n:
def cal(w, x):
  sum = 0
  sum = float(sum)
  for i in range(11):
    sum += w[i]*x[i]
  return sum

#some initialization for N and x_n:
#N
N = 256
#xn
file = open('hw1_train.txt', 'r')
data = []
for line in file:
  number_strings = line.split()
  numbers = [float(n) for n in number_strings]
  data.append(numbers)

#we have x_0 = 1 and M = 4N
#initialize x_0 and add it to every x_n:
x0 = 1
for row in data:
    row.insert(0, x0)
data_n = numpy.array(data)

#run the experiment with the conditions to get the median number of updates:
updates = []
for E in range(1000):
    #w0 initialization
    w = [0]*11
    #M iteration
    M = int(4*N)
    cnt = 0
    update = 0
    while True:
        #random pick a xn
        x = randint(0, 255)
        #calculate the wfxn
        num = cal(w, data_n[x])
        #check sign
        if (num >= 0 and data_n[x][11] < 0) or (num < 0 and data_n[x][11] > 0):
            for j in range(11):
```

```
                w[j] = w[j] + data_n[x][j]*data_n[x][11]
            cnt = 0
            update += 1
        else:
            cnt += 1
        if cnt >= M: #stop at M correct consecutive check
            break
    #put this round's update into the updates list
    updates.append(update)

#get the medain number of upadtes
print(statistics.median(updates))
```

I get **median** number of updates = $446.0 \approx 400$

16.

Ans: (e)

source code:

```
#import libraries:
import numpy
from numpy import loadtxt
import random
from random import randint
import statistics
import math

#define the function to calculate w_f x_n:
def cal(w, x):
  sum = 0
  sum = float(sum)
  for i in range(11):
    sum += w[i]*x[i]
  return sum

#some initialization for N and x_n:
#N
N = 256
#xn
file = open('hw1_train.txt', 'r')
data = []
for line in file:
  number_strings = line.split()
  numbers = [float(n) for n in number_strings]
  data.append(numbers)

#we have x_0 = 1 and M = 4N
#initialize x_0 and add it to every x_n:
x0 = 1
for row in data:
    row.insert(0, x0)
data_n = numpy.array(data)

#run the experiment with the conditions to get the median of w_0:
w0s = []
for E in range(1000):
    #w0 initialization
    w = [0]*11
    #M iteration
    M = int(4*N)
    cnt = 0
    while True:
        #random pick a xn
        x = randint(0, 255)
        #calculate the wfxn
        num = cal(w, data_n[x])
        #check sign
        if (num >= 0 and data_n[x][11] < 0) or (num < 0 and data_n[x][11] > 0):
            for j in range(11):
                w[j] = w[j] + data_n[x][j]*data_n[x][11]
```

```
            cnt = 0
        else:
            cnt += 1
        if cnt >= M: #stop at M correct consecutive check
            break
    #put this round's wf's w0 into the w0s list
    w0s.append(float(w[0]))

#get the medain number of w0
print(statistics.median(w0s))
```

I get **median** number of $w_0 = 34.0 \approx 40$

17.

Ans: (d)

source code:

```
#import libraries:
import numpy
from numpy import loadtxt
import random
from random import randint
import statistics
import math

#define the function to calculate w_f x_n:
def cal(w, x):
  sum = 0
  sum = float(sum)
  for i in range(11):
    sum += w[i]*x[i]
  return sum

#some initialization for N and x_n:
#N
N = 256
#xn
file = open('hw1_train.txt', 'r')
data = []
for line in file:
  number_strings = line.split()
  numbers = [float(n) for n in number_strings]
  data.append(numbers)

#we have x_0 = 1 and M = 4N
#initialize x_0 and add it to every x_n:
x0 = 1
for row in data:
    row.insert(0, x0)
data_n = numpy.array(data)

#scale  down each x_n by 2:
for rows in data_n:
    rows[:11] *= 0.5

#run the experiment with the conditions to get the median number of updates:
updates = []
for E in range(1000):
    #w0 initialization
    w = [0]*11
    #M iteration
    M = int(4*N)
    cnt = 0
    update = 0
    while True:
        #random pick a xn
        x = randint(0, 255)
        #calculate the wfxn
```

```
        num = cal(w, data_n[x])
        #check sign
        if (num >= 0 and data_n[x][11] < 0) or (num < 0 and data_n[x][11] > 0):
            for j in range(11):
                w[j] = w[j] + data_n[x][j]*data_n[x][11]
            cnt = 0
            update += 1
        else:
            cnt += 1
        if cnt >= M: #stop at M correct consecutive check
            break
    #put this round's update into the updates list
    updates.append(update)

#get the medain number of upadtes
print(statistics.median(updates))
```

I get **median** number of updates = $452.5 \approx 400$

18.

Ans: (d)

source code:

```
#import libraries:
import numpy
from numpy import loadtxt
import random
from random import randint
import statistics
import math

#define the function to calculate w_f x_n:
def cal(w, x):
  sum = 0
  sum = float(sum)
  for i in range(11):
    sum += w[i]*x[i]
  return sum

#some initialization for N and x_n:
#N
N = 256
#xn
file = open('hw1_train.txt', 'r')
data = []
for line in file:
  number_strings = line.split()
  numbers = [float(n) for n in number_strings]
  data.append(numbers)

#we have x_0 = 0 and M = 4N
#initialize x_0 and add it to every x_n:
x0 = 0
for row in data:
    row.insert(0, x0)
data_n = numpy.array(data)

#run the experiment with the conditions to get the median number of updates:
updates = []
for E in range(1000):
    #w0 initialization
    w = [0]*11
    #M iteration
    M = int(4*N)
    cnt = 0
    update = 0
    while True:
        #random pick a xn
        x = randint(0, 255)
        #calculate the wfxn
        num = cal(w, data_n[x])
        #check sign
        if (num >= 0 and data_n[x][11] < 0) or (num < 0 and data_n[x][11] > 0):
            for j in range(11):
```

```
                w[j] = w[j] + data_n[x][j]*data_n[x][11]
            cnt = 0
            update += 1
        else:
            cnt += 1
        if cnt >= M: #stop at M correct consecutive check
            break
    #put this round's update into the updates list
    updates.append(update)

#get the medain number of upadtes
print(statistics.median(updates))
```

I get **median** number of updates = $451.5 \approx 400$

19.

Ans: (e)

source code:

```
#import libraries:
import numpy
from numpy import loadtxt
import random
from random import randint
import statistics
import math

#define the function to calculate w_f x_n:
def cal(w, x):
  sum = 0
  sum = float(sum)
  for i in range(11):
    sum += w[i]*x[i]
  return sum

#some initialization for N and x_n:
#N
N = 256
#xn
file = open('hw1_train.txt', 'r')
data = []
for line in file:
  number_strings = line.split()
  numbers = [float(n) for n in number_strings]
  data.append(numbers)

#we have x_0 = -1 and M = 4N
#initialize x_0 and add it to every x_n:
x0 = -1
for row in data:
    row.insert(0, x0)
data_n = numpy.array(data)

#run the experiment with the conditions and get the median number of w_0*x_0:
w0x0s = []
for E in range(1000):
    #w0 initialization
    w = [0]*11
    #M iteration
    M = int(4*N)
    cnt = 0
    while True:
        #random pick a xn
        x = randint(0, 255)
        #calculate the wfxn
        num = cal(w, data_n[x])
        #check sign
        if (num >= 0 and data_n[x][11] < 0) or (num < 0 and data_n[x][11] > 0):
            for j in range(11):
                w[j] = w[j] + data_n[x][j]*data_n[x][11]
```

```
            cnt = 0
        else:
            cnt += 1
        if cnt >= M: #stop at M correct consecutive check
            break
    #put this round's w0*x0 to the list
    w0x0s.append(float(w[0]*x0))

#get the medain number of upadtes
print(statistics.median(w0x0s))
```

I get **median** number of $w_0 \cdot x_0 = 34.0 \approx 40$

20.

Ans: (c)

source code:

```
#import libraries:
import numpy
from numpy import loadtxt
import random
from random import randint
import statistics
import math

#define the function to calculate w_f x_n:
def cal(w, x):
  sum = 0
  sum = float(sum)
  for i in range(11):
    sum += w[i]*x[i]
  return sum

#some initialization for N and x_n:
#N
N = 256
#xn
file = open('hw1_train.txt', 'r')
data = []
for line in file:
  number_strings = line.split()
  numbers = [float(n) for n in number_strings]
  data.append(numbers)

#we have x_0 = 0.1126 and M = 4N
#initialize x_0 and add it to every x_n:
x0 = 0.1126
for row in data:
    row.insert(0, x0)
data_n = numpy.array(data)

#run the experiment with the conditions and get the median number of w_0*x_0:
w0x0s = []
for E in range(1000):
    #w0 initialization
    w = [0]*11
    #M iteration
    M = int(4*N)
    cnt = 0
    while True:
        #random pick a xn
        x = randint(0, 255)
        #calculate the wfxn
        num = cal(w, data_n[x])
        #check sign
        if (num >= 0 and data_n[x][11] < 0) or (num < 0 and data_n[x][11] > 0):
            for j in range(11):
                w[j] = w[j] + data_n[x][j]*data_n[x][11]
```

```
            cnt = 0
        else:
            cnt += 1
        if cnt >= M: #stop at M correct consecutive check
            break
    #put this round's w0*x0 to the list
    w0x0s.append(float(w[0]*x0))

#get the medain number of upadtes
print(statistics.median(w0x0s))
```

I get **median** number of $w_0 \cdot x_0 = 0.4310778400000001 \approx 0.4$