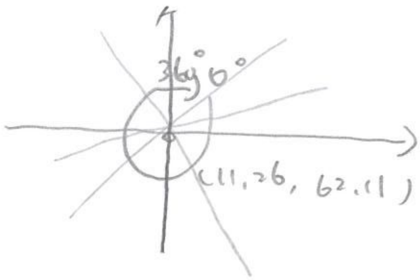
Machine Learning HW2 R10922A16 蔡家家

1. $h(x) = \text{sign}(w^T x)$ such that $w_1(x_1 - 11.26) + w_2(x_2 - 62.11) = -w_0$

we can assume $w_0 = 0$ to form the following graph



those perceptions will pass (11.26, 62.11) always.

so basically we only consider its angle.

we can map these angle to a 1D line.

which can reduce this problem into pos/neg ray

the positive - negative ray's growth function is $\underline{m_H(N) = 2N}$ # $(N+1 + N+1 - 2)$ $\begin{matrix}xx\\oo\end{matrix}$

[d] #

2. Since we have 1126 perceptions, we have a hypothesis set with size $\underline{1126}$

the upper bound of VC Dimension will be $2^{Vcd} = 1126$

$\Rightarrow$ $\underline{VcD = \log_2(1126)}$ #

[a] #

3. [a] $V_cD = 4$ $\xrightarrow{\;-1\;|\;+1\;|\;-1\;|\;+1\;|\;-1\;}$

the $+1, -1, +1, -1, +1$ case is a 5 inputs case cannot be shattered.

$\Rightarrow$ VC Dimension ≤ 4

o : +1 x : -1

oooo
ooox
xooo
oxoo
ooxo
ooxx
xoox
xxoo
oxxo
xoxo
oxox
oxxx
xoxx
xxox
xxxo

here is all the case when we have 4 inputs, every one can find proper position, so 4 inputs can be shattered.

VC Dimension ≥ 4

$\Rightarrow$ $\underline{\text{VC Dimension} = 4}$

3-[b] VC Dimension $= 4$.

a degree n polynomial has at most n roots, which means you cannot find patterns like $- + - + - + \cdots$ or $+ - + - + - \cdots$ with length $n+2$

$\Rightarrow$ VC Dimension $\leq n+1$

for any set of $(x_1, y_1), (x_2, y_2) \dots, (x_{n+1}, y_{n+1})$, we can interpolate them all, so when $y_i = \pm 1$, any set of $n+1$ points shattered.

$\Rightarrow$ VC Dimension $\geq n+1$

$\Rightarrow$ $\underline{\text{VC Dimension} = n+1 = 3+1 = 4}$

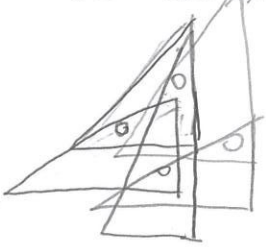
(c) when $\sin(wx) > x$, $y = +1$

$\Rightarrow$ we have $h(x) = \text{sign}(\sin(wx) - x)$

basically, sin family has VC Dimension $= \infty$

in this case, we can find 4 inputs shattered, $\underline{\text{VC Dimension} \geq 4}$

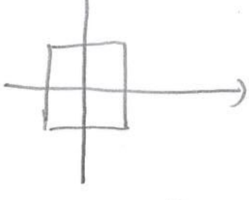
[d]



we can use the following 4 points to shattered.

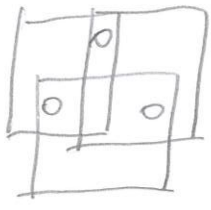
$\underline{\text{VC Dimension} \geq 4}$

[e]



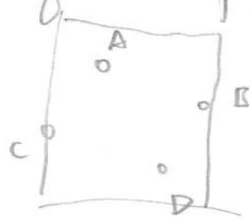
We can shatter 3 points

VC Dimension ≥ 3.



any set of 4 points cannot be shattered

suppose we have the following 4 inputs, in this case, if we want to contain B and C, we will always take one of A and D, which means we cannot shattered at 4 points.

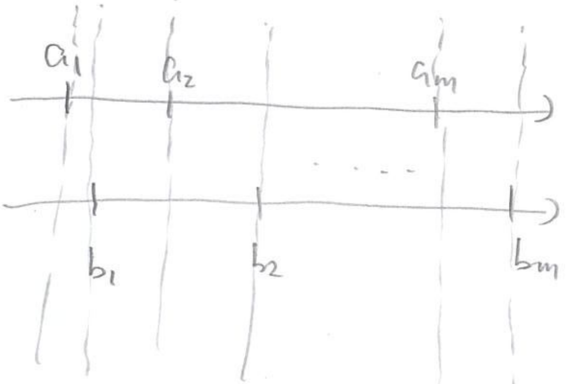


VC Dimension ≤ 3

$\Rightarrow$ VC Dimension $= 3$

$\underline{\text{The smallest one is (e)}}$ #

4. we can construct a line and b line seperately like below.



if we combine these two line, we can form a M union of positive intervals.

To prove VC Dimension $\geq 2M$ we can use these $2M+1$ intervals to compose all the answer, the most extreme condition is to use $2M$ intervals to construct answer like $\underbrace{-1, +1, -1, +1 \cdots +1}_{2M}$ or $\underbrace{+1, -1, +1, -1, \cdots -1}_{2M}$, other case will use less intervals than this condition.

VC Dimension $\geq 2M$

To prove VC Dimension $\leq 2M$, we can form a condition $\underbrace{+1, -1, +1, \cdots -1}_{2M}$ $\underset{\text{isolated } +1}{+1}$ which the rightmost $+1$ cannot find a proper position.

$2M+1$ points cannot be shattered.

VC Dimension $\leq 2M$. $\Rightarrow$ VC Dimension $= 2M$. [b]

The VC Dimension $\geq 2M$.

we can define $X = \begin{bmatrix} x_1^T \\ \vdots \\ x_{2M} \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & \cdots & 0 \\ 1 & 1 & & 0 \\ 1 & 0 & \ddots & \vdots \\ \vdots & & & 0 \\ 1 & & 0 & 1 \end{bmatrix}}_{(I)_{2M \times 2M}}$ $2M$ which is an invertible matrix

$\Rightarrow Xw = y \Leftrightarrow w = X^{-1}y$. $y = \begin{bmatrix} +1 \\ -1 \\ +1 \\ \vdots \\ -1 \end{bmatrix}$ VC D $\geq 2M$

for VC Dimension $\leq 2M$ $X = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 1 & 1 & & 0 \\ 1 & 0 & & \\ 1 & 0 & 1 & 1 \end{bmatrix}_{2M+1}$ $w^T x_{2M+1} = w^T x_{2M} + w^T x_{2M-1} - w^T x_1$.

this x have dependence

We have $2M$ parameter, $d_{vc} = 2M$.
(degree of freedom)

5. necessary conditions for $d_{vc}(H) \le d$

Some set of d distinct inputs is shattered by H.

Some set of $d+1$ distinct inputs is not shattered by H.

any set of $d+1$ distinct inputs is not shattered by H.

[C]

6. $h(x) = wx$

$$\Rightarrow E_{in}(w) = \frac{1}{N}\sum_{n=1}^{N}(h(x_n) - y_n)^2$$

To minimize, take gradient.

$$\frac{\partial E_{in}(w)}{\partial w} = \cancel{\frac{2}{N}}\sum_{n=1}^{N}(wx_n - y_n)\cdot x_n = 0$$

$$\Rightarrow w\sum_{n=1}^{N}x_n = \sum_{n=1}^{N}y_n x_n$$

$$\Rightarrow w = \frac{\sum_{n=1}^{N}y_n x_n}{\sum_{n=1}^{N}x_n}$$

(b)

7. we can solve these problem by replace the $h(x)$ in likelihood function by $p(x)$

likelihood function $= \prod_{n=1}^{N} h(y_n x_n)$

[a] $\prod_{n=1}^{N}\frac{e^{-\lambda}\lambda^{x_n}}{x_n!} \xrightarrow{\text{take } \ln} \ln\left(\prod_{n=1}^{N}\frac{e^{-\lambda}\lambda^{x_n}}{x_n!}\right) \Rightarrow \frac{1}{N}\sum_{n=1}^{N}\ln(e^{-\lambda}\lambda^{x_n}) - \ln(x$

$$\Rightarrow \frac{1}{N}\sum_{n=1}^{N}(-\lambda + x_n\ln\lambda - \ln(x_n!)) \Rightarrow \frac{\partial(\ )}{\partial\lambda} = 0 \Rightarrow \frac{1}{N}\sum_{n=1}^{N}(-1 + \frac{x_n}{\lambda}) = 0$$

$-N$

$$\Rightarrow \sum_{n=1}^{N}\frac{x_n}{\lambda} = 1 \Rightarrow \frac{1}{N}\sum_{n=1}^{N}x_n = \frac{1}{N}\sum_{n=1}^{N}\lambda \Rightarrow \lambda = \frac{1}{N}\sum_{n=1}^{N}x_n = \bar{x}$$

[b] $p(x) = \frac{1}{\sqrt{2\pi}}e^{\frac{-(x_n-\mu)^2}{2}}$

$$\prod_{n=1}^{N}\frac{1}{\sqrt{2\pi}}e^{\frac{-(x_n-\mu)^2}{2}} \Rightarrow \left(\frac{1}{\sqrt{2\pi}}\right)^N e^{\sum_{n=1}^{N}\frac{-1}{2}(x_n-\mu)^2} \xrightarrow{\text{take } \ln} N\ln\left(\frac{1}{\sqrt{2\pi}}\right) + \left(\sum_{n=1}^{N}\frac{-1}{2}(x_n-\mu)^2\right)$$

$$\frac{\partial(\ )}{\partial\mu} = 0 \Rightarrow \sum_{n=1}^{N}\frac{-1}{\cancel{2}}\times(x_n-\mu)'(x_n-\mu)\times\cancel{2} = 0 \Rightarrow \sum_{n=1}^{N}x_n = N\mu$$

$$\Rightarrow \mu = \frac{1}{N}\sum_{n=1}^{N}x_n = \bar{x}$$

[C] $\prod_{n=1}^{N} \frac{1}{2} e^{-|x_n-\mu|} \xRightarrow{\text{take } \ln} \sum_{n=1}^{N} \ln(\frac{1}{2}) - |x_n-\mu|$

$\Rightarrow \frac{\partial(\ )}{\partial \mu} \Rightarrow \sum_{n=1}^{N} \frac{x_n-\mu}{|x_n-\mu|} = 0$, μ is the median of x_n to obtain the maximum likelihood instead of $\bar{x} = \text{mean}(x_1, \cdots x_n)$

[d]

$\prod_{n=1}^{N} (1-\theta)^{x_n-1}\theta \Rightarrow (1-\theta)^{\sum_{n=1}^{N} x_n - N} \theta^N \xRightarrow{\text{take } \ln} \left(\sum_{n=1}^{N} x_n - N\right)(1-\theta) + N\ln\theta$

$\Rightarrow \frac{\partial(\ )}{\partial\theta} = 0 \Rightarrow \frac{1}{\theta}N + (1-\theta)'\left(\sum_{n=1}^{N}(x_n) - N\right)\frac{1}{1-\theta} = 0 \Rightarrow \frac{N}{\theta} = \frac{\sum_{n=1}^{N}(x_n) - N}{1-\theta}$

$\Rightarrow N - \cancel{N\theta} = \sum_{n=1}^{N} x_n \cdot \theta - \cancel{N\theta} \Rightarrow \theta = \frac{N}{\sum_{n=1}^{N} x_n} = \frac{1}{\bar{x}}$

[C] is the answer, μ is the median of x_n instead of mean of x_n

8. We can replace the original function by $h_w = \frac{1+w^Tx+|w^Tx|}{2+2|w^Tx|}$

$\Rightarrow E_{in} = \frac{-1}{N}\ln\left(\frac{1+y_nw^Tx_n+|y_nw^Tx_n|}{2+2|y_nw^Tx_n|}\right)$

let $\bigcirc = \left(\frac{1+y_nw^Tx_n+|y_nw^Tx_n|}{2+2|y_nw^Tx_n|}\right)$ and $\square = y_nw^Tx_n$

$\nabla E_{in}(w) = \frac{\partial E_{in}}{\partial w}$.

$\Rightarrow = \frac{-1}{N}\sum_{n=1}^{N}\left(\frac{\partial(\ln\bigcirc)}{\partial\bigcirc}\right)\left(\frac{\partial\left(\frac{1+\square+|\square|}{2+2\square}\right)}{\partial\square}\right)\left(\frac{\partial(y_nw^Tx_n)}{\partial w_i}\right)$

$\Rightarrow = \frac{-1}{N}\sum_{n=1}^{N}\left(\frac{1}{\bigcirc}\right)\left(\frac{(1+\square+|\square|)'(2+2|\square|)-(1+\square+|\square|)(2+2|\square|)'}{(2+2|\square|)^2}\right)(y_nx_{n,i})$

$\Rightarrow = \frac{-1}{N}\sum_{n=1}^{N}\left(\frac{1}{\bigcirc}\right)\left(\frac{(1+\frac{\square}{|\square|})(2+2|\square|)-(1+\square+|\square|)(2\frac{\square}{|\square|})}{(2+2|\square|)^2}\right)(y_nx_{n,i})$

$\Rightarrow = \frac{-1}{N}\sum_{n=1}^{N}\left(\frac{1}{\bigcirc}\right)\left(\frac{(2+\cancel{2\frac{\square}{|\square|}}+2|\square|+\cancel{2\square})-(\cancel{2\frac{\square}{|\square|}}+2\frac{\square^2}{|\square|}+\cancel{2\square})}{(2+2|\square|)^2}\right)(y_nx_{n,i})$

$\Rightarrow = \frac{-1}{N}\sum_{n=1}^{N}\left(\frac{1}{\bigcirc}\right)\left(\frac{2+2|\square|-2\frac{\square^2}{|\square|}}{(2+2|\square|)^2}\right)(y_nx_{n,i})$

$\Rightarrow = \frac{-1}{N}\sum_{n=1}^{N}\left(\frac{2+2|\square|}{1+\square+|\square|}\right)\left(\frac{\cancel{2}+2|\square|-\cancel{2|\square|}}{(\cancel{2}+2|\square|)^{\cancel{2}}}\right)(y_nx_{n,i})$

$\Rightarrow = \frac{-1}{N}\sum_{n=1}^{N}\frac{1}{(1+\square+|\square|)(1+|\square|)}(y_nx_{n,i})$

$\Rightarrow \frac{-1}{N}\sum_{n=1}^{N}\frac{y_nx_{n,i}}{(1+y_nX^TX_n+|y_nX^TX_n|)(1+|y_nX^TX_n|)}$ # [a] #

9.

$$E_{in}(w) = \frac{1}{N} \sum_{n=1}^{N} \| Xw - y \|^2$$ (from lecture)

$$\nabla E_{in}(w) = \frac{2}{N} X^T (Xw - y)$$

$$\nabla^2 E_{in}(w) = \frac{2}{N} (X^T X)$$ [b] #

$u = (X^T X)^{-1} \nabla E_{in}(w)$

10.

$$W_{t+1} = \cancel{W_t} - (X^T X)^{-1} (\cancel{X^T X w_t} - X^T y)$$

$$= (X^T X)^{-1} X^T y$$ constant term

means only needs one step, and no matter the initial w_0

[a] #.

11.

$\epsilon = 0.05$, $\delta = 0.1$. $d_{vc} = 2$

$$P_D [\, | E_{in}(g) - E_{out}(g) | > \epsilon \,] \le 4 (2N)^{d_{vc}} e^{(\frac{-1}{8} \epsilon^2 N)}$$

we want to make sure the bound.

$\Rightarrow$ let $4 (2N)^2 e^{(\frac{-1}{8}(0.05)^2 N)} \le 0.1$

$$\Rightarrow N^2 e^{(\frac{-1}{8}(0.05)^2 N)} \le \frac{1}{160}$$

⇓ take ln

$$\Rightarrow 2\ln N + (\frac{-1}{8}(0.05)^2 N) \le \ln(\frac{1}{160})$$

⇓ a little bit hard to solve.

put answer into it to solve.

[a] $2\ln 100 + (\frac{-1}{8}(0.05)^2 \times 100) \doteq 9.17\cdots > \ln(\frac{1}{160}) \doteq -5.0751\ldots$

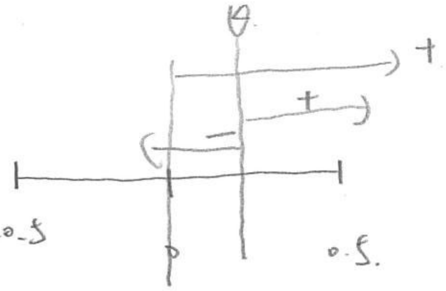
[b] $2\ln 1000 + (\frac{-1}{8}(0.05)^2\, 1000) \doteq 13.503\ldots > \ln(\frac{1}{160})$

[c] $2\ln 10000 + (\frac{-1}{8}(0.05)^2\, 10000) \doteq 15.2957\cdots > \ln(\frac{1}{160})$

[d] $2\ln 100000 + (\frac{-1}{8}(0.05)^2\, 100000) \doteq -8.224\cdots < \ln(\frac{1}{160})$ # $N = 100000$ # [d]

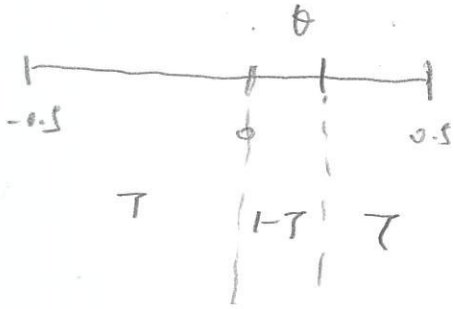
[e] $2\ln 1000000 + (\frac{-1}{8}(0.05)^2\, 1000000) \doteq -284.869\cdots < \ln(\frac{1}{160})$

12.



already know $s=+1$, assume $\theta>0$.

the noise τ is uniform



$[0,\theta]$ $E_{out} = 1-\tau$

$[-0.5,0]$ and $[\theta, 0.5]$ $E_{out} = \tau$

$E_{out}(h,\tau) = |\theta|(1-\tau) + (1-|\theta|)\tau$

$\Rightarrow$ but $|\theta|$ might exceed the border.

replace $|\theta|$ by $\min(|\theta|, 0.5)$

we have

$$E_{out} = \min(|\theta|, 0.5)(1-\tau) + (1-\min(|\theta|, 0.5))\tau$$

$$\Rightarrow E_{out} = \min(|\theta|, 0.5) - \tau\min(|\theta|, 0.5) + \tau - \min(|\theta|, 0.5)\tau$$

$$\Rightarrow E_{out} = \min(|\theta|, 0.5)(1-2\tau) + \tau$$

[d]

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

@dataclass
class DecisionStump:
    art_data: bool
    tau: Optional[float] = None
    n_init: Optional[int] = None
    n: Optional[int] = None

    def generate_data(self, n, tau):
        x = np.random.uniform(low=-0.5, high=0.5, size=n)
        y = [self.sign(value) for value in x]
        for idx, label in enumerate(y):
            if np.random.random() < tau:
                y[idx] = -label

        return np.array(x), np.array(y)

    def fit(self, x, y):
        self.ans_1d = []
        self.ans_multi_d = []
        # self.test = []

        if self.art_data:
            # x_out, y_out = self.generate_data(100000, self.tau)
            # x_out = np.reshape(x_out, (len(x_out), 1))
            # y_out = np.reshape(y_out, (len(y_out), 1))
            for i in range(self.n_init):
                # artificial data
                self.x, self.y = self.generate_data(self.n, self.tau)
                self.x = np.reshape(self.x, (len(self.x), 1))
                self.y = np.reshape(self.y, (len(self.y), 1))

                # sample number and dimension of training data
                self.n = self.x.shape[0]
                self.d = self.x.shape[1]

                if self.d == 1:
                    ans1, ans2, ans3, ans4 = self.run_DS_1D(self.x, self.y)
                    self.ans_1d += [ans1]
                    # self.test += [self.error(x_out, y_out, ans4, ans2) - ans3]

                else:
                    # won't use in this hw
                    self.ans_multi_d += self.run_DS_multiD(self.x, self.y)
        else:
            self.x = np.array(x)
            self.y = np.array(y)

            # sample number and dimension of training data
            self.n = self.x.shape[0]
            self.d = self.x.shape[1]

            result = self.run_DS_multiD(self.x, self.y)

            self.best_of_best_ein = result[0]
            self.worst_of_best_ein = result[1]
            self.best_of_best_theta = result[2]
            self.worst_of_best_theta = result[3]
            self.best_of_best_s = result[4]
            self.worst_of_best_s = result[5]
            self.best_of_best_dim = result[6]
            self.worst_of_best_dim = result[7]

        return self

    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 run_DS_1D(self, x, y):
        x_n_sorted = sorted(x)
        ninf = -1e15
        thetas = np.array([ninf])
        # generate theta
        for i in range(self.n-1):
            thetas = np.append(thetas, (x_n_sorted[i]+x_n_sorted[i+1])/2)
        thetas = np.array(thetas)

        direction = [1, -1]
        e_best = 1
        theta_best = 0

        # calculate error and record best
        for s in direction:
            for theta in thetas:
                wrong = self.error(x, y, s, theta)
                if wrong < e_best:
                    e_best = wrong
                    theta_best = theta
                    s_best = s

        if self.art_data:
            e_out = self.error_out(theta_best)
        else:
            # dummy eout, when we use real data, eout can be computed from test data
            e_out = 1
        e_in = e_best

        return e_out - e_in, theta_best, e_in, s_best

    def run_DS_multiD(self, x, y):
        best_thetas = []
        best_eins = []
        best_ss = []

        y = np.reshape(y, (len(y), 1))

        for dim in range(self.d):
            x_dim = np.reshape(x[:,dim], (len(x[:,dim]), 1))
            result = self.run_DS_1D(x_dim, y)
            best_thetas += [result[1]]
            best_eins += [result[2]]
            best_ss += [result[3]]

        best_of_best_theta = best_thetas[np.argmin(best_eins)]
        worst_of_best_theta = best_thetas[np.argmax(best_eins)]

        best_of_best_ein = best_eins[np.argmin(best_eins)]
        worst_of_best_ein = best_eins[np.argmax(best_eins)]

        best_of_best_s = best_ss[np.argmin(best_eins)]
        worst_of_best_s = best_ss[np.argmax(best_eins)]

        best_of_best_dim = np.argmin(best_eins)
        worst_of_best_dim = np.argmax(best_eins)

        return best_of_best_ein, worst_of_best_ein, best_of_best_theta, worst_of_best_theta,
best_of_best_s, worst_of_best_s, best_of_best_dim, worst_of_best_dim

    def error_out(self, theta):
        return min(np.abs(theta), 0.5) * (1 - 2 * self.tau) + (self.tau)

    def predict(self, x, y, best=None):
        predict_label = []
        wrong = 0
        if not best:
            dim = self.worst_of_best_dim
            s = self.worst_of_best_s
            x_dim = np.reshape(x[:,dim], (len(x), 1))
            for idx, data in enumerate(x_dim):
                predict_label += [int(s*self.sign(data-self.worst_of_best_theta))]
                if y[idx] != s*self.sign(data-self.worst_of_best_theta):
                    wrong += 1
                eout = wrong/len(y)
        else:
            dim = self.best_of_best_dim
            s = self.best_of_best_s
            x_dim = np.reshape(x[:,dim], (len(x), 1))
            for idx, data in enumerate(x_dim):
                predict_label += [s*self.sign(data-self.best_of_best_theta)]
                if y[idx] != s*self.sign(data-self.best_of_best_theta):
                    wrong += 1
                eout = wrong/len(y)

        return predict_label, eout

    def sign(self, value):
        """
        The sign function

        Args:
            value (float): the dot value from PLA iteration

        Returns:
            int: if value > 0 will be 1 otherwise -1
        """
        if value > 0:
            return 1
        else:
            return -1
```

In [ ]:
```
import numpy as np
from DecisionStump import DecisionStump
```

In [ ]:
```
# dummy data to init Decision Stump when using artificial data
dummy_x = 0
dummy_y = -1
```

In [ ]:
```
p13_kwargs = {
    'tau': 0,
    'art_data': True,
    'n_init': 10000,
    'n': 2
}

DS_p13 = DecisionStump(**p13_kwargs).fit(dummy_x, dummy_y)
ans_p13 = np.mean(DS_p13.ans_1d)
print(ans_p13)
# print(np.mean(DS_p13.test))
```

```
0.29388650380312675
```

In [ ]:
```
p14_kwargs = {
    'tau': 0,
    'art_data': True,
    'n_init': 10000,
    'n': 128
}

DS_p14 = DecisionStump(**p14_kwargs).fit(dummy_x, dummy_y)
ans_p14 = np.mean(DS_p14.ans_1d)
print(ans_p14)
# print(np.mean(DS_p14.test))
```

```
0.003910929152214343
```

In [ ]:
```
p15_kwargs = {
    'tau': 0.2,
    'art_data': True,
    'n_init': 10000,
    'n': 2
}

DS_p15 = DecisionStump(**p15_kwargs).fit(dummy_x, dummy_y)
ans_p15 = np.mean(DS_p15.ans_1d)
print(ans_p15)
# print(np.mean(DS_p15.test))
```

```
0.3905382425955801
```

In [ ]:
```
p16_kwargs = {
    'tau': 0.2,
    'art_data': True,
    'n_init': 10000,
    'n': 128
}

DS_p16 = DecisionStump(**p16_kwargs).fit(dummy_x, dummy_y)
ans_p16 = np.mean(DS_p16.ans_1d)
```

```
print(ans_p16)
# print(np.mean(DS_p16.test))
```

```
0.013694312914973472
```

In [ ]:
```
"""
load data and preprocess
"""
# read data
with open('hw2_train.dat', 'rb') as f:
    training_data = np.array([np.float64(i.split()) for i in f.readlines()])

# turn x and y into numpy array
# x is the input feature vector space and y is the corresponding label
x = np.array(training_data[:,0:10])
y = np.reshape(np.array(list(map(int, training_data[:,10]))), (len(x), 1))
print(x.shape)
print(y.shape)

N = len(x)

with open('hw2_test.dat', 'rb') as f:
    test_data = np.array([np.float64(i.split()) for i in f.readlines()])

# turn x and y into numpy array
# x is the input feature vector space and y is the corresponding label
x_test = np.array(test_data[:,0:10])
y_test = np.reshape(np.array(list(map(int, test_data[:,10]))), (len(x_test), 1))
print(x_test.shape)
print(y_test.shape)

N = len(x)
```

```
(192, 10)
(192, 1)
(64, 10)
(64, 1)
```

In [ ]:
```
"""
build p17~p20 model
"""
real_data_kwargs = {
    'art_data': False
}
DS_remaining_question = DecisionStump(**real_data_kwargs).fit(x, y)
```

In [ ]:
```
ans_p17 = DS_remaining_question.best_of_best_ein
ans_p17
```

Out[ ]:
```
0.026041666666666668
```

In [ ]:
```
best_eout = DS_remaining_question.predict(x_test, y_test, True)[1]
ans_p18 = best_eout
ans_p18
```

Out[ ]:
```
0.078125
```

In [ ]:
```
best_ein = DS_remaining_question.best_of_best_ein
worst_ein = DS_remaining_question.worst_of_best_ein
```

```
worst_eout = DS_remaining_question.predict(x_test, y_test, False)[1]

ans_p19 = worst_ein - best_ein
ans_p19
```

Out[ ]: 0.30208333333333333

In [ ]:
```
ans_p20 = worst_eout - best_eout
ans_p20
```

Out[ ]: 0.34375