ML HW1

蔡家豪 R10922A16

March 2023

1 Basics of Machine Learning

1. [a]
I think [a] is one of the problems that Machine Learning techniques can solve. In the real world, we can now use some ML methods to transform speech into text and feed those texts to a Large Language Model such as ChatGPT to obtain the answers. We can see the input vector space is $\vec{x}$, and $\vec{x}$ is the user's request, and try to approximate f by g, f is an unknown function that can reflect good answers. We can use some evaluation methods to make g close to f. This is what I think [a] is the best fit for ML to solve.
2. [d]
Assume Z is the learning rate which used to update W_{t+1}
now we have
$W_{t+1} \leftarrow W_t + y_{n(t)}x_{n(t)} \cdot Z$
if we want to make sure W_{t+1} is correct on sample $(y_{n(t)}, x_{n(t)})$, then we need to let $W_{t+1}y_{n(t)}x_{n(t)} > 0$
We can replace W_{t+1} by $W_t + y_{n(t)}x_{n(t)} \cdot Z$ to derive the answer we want. Below is the procedure:

$W_{t+1}x_{n(t)}y_{n(t)} = (W_t + x_{n(t)}y_{n(t)} \cdot Z)x_{n(t)}y_{n(t)} > 0$
$\Rightarrow ||x_{n(t)}||^2 y_{n(t)}^2 \cdot Z > -y_{n(t)}W_t^\intercal x_{n(t)}$
$\Rightarrow Z > \frac{-W_t^\intercal x_{n(t)}y_{n(t)}}{||x_{n(t)}||^2 y_{n(t)}^2} = \frac{-W_t^\intercal x_{n(t)}}{||x_{n(t)}||^2 y_{n(t)}}$
We know $y_{n(t)} \in \{-1, 1\}$ so $y_{n(t)}^2 = 1$

we can rewrite the equation into
$Z > \frac{-y_{n(t)}^2 W_t^\intercal x_{n(t)}}{y_{n(t)}||x_{n(t)}||^2} = \frac{-y_{n(t)}W_t^\intercal x_{n(t)}}{||x_{n(t)}||^2}$
We want to make sure $Z > \frac{-y_{n(t)}W_t^\intercal x_{n(t)}}{||x_{n(t)}||^2}$ so [d]$W_{t+1} \leftarrow W_t + y_{n(t)}x_{n(t)} \cdot \lfloor \frac{-y_{n(t)}W_t^\intercal x_{n(t)}}{||x_{n(t)}||^2} + 1 \rfloor$ is the only answer that can make sure the learning rate will be larger and not equal to $\frac{-y_{n(t)}W_t^\intercal x_{n(t)}}{||x_{n(t)}||^2}$
3. [c]
We have $z_n = \frac{x_n}{||x_n||}, \rho_z = \min_n \frac{y_n w_f^\intercal z_n}{||x_n||}$

and replace the ρ by ρ_z on lecture 1 page 41
$w_f^\intercal w_1 \geq w_f^\intercal w_0 + \rho_z$
$w_f^\intercal w_2 \geq w_f^\intercal w_1 + \rho_z$
$\vdots \quad \vdots \quad \vdots$
$w_f^\intercal w_T \geq w_f^\intercal w_{T-1} + \rho_z$
$\Rightarrow w_f^\intercal w_T \geq w_f^\intercal w_0 + T\rho_z$
After normalization $\underset{n}{max}\,||x_n||^2$ will become $\underset{n}{max}\,||z_n||^2 = 1$, the chain rule can be rewritten into:
$|||w_1|^2 \leq |||w_0|^2 + 1$
$|||w_2|^2 \leq |||w_1|^2 + 1$
$\vdots \quad \vdots \quad \vdots$
$|||w_T|^2 \leq |||w_{T-1}|^2 + 1$
$\Rightarrow |||w_T|^2 \leq |||w_0|^2 + T$

now we can rewrite the formula to find the new bound.
$1 \geq \frac{w_f^\intercal w_T}{||w_f||||w_T||} \geq \frac{T \cdot \rho_z}{1\sqrt{T} \cdot 1}$
$\Rightarrow \sqrt{T} \leq \frac{1}{\rho_z} \Rightarrow T \leq \frac{1}{\rho_z^2}$
so the answer is $[c]\frac{1}{\rho_z^2}$
4. $[b]$
We have a new bound U in problem 3, the difference between U and U_{orig} are ρ, ρ_z, and R. $\rho_z = \underset{n}{min}\,\frac{y_n w_f^\intercal x_n}{||w_f||||x_n||}$, if we assume $||w_f||$ is 1, we will have the following relation:
$\rho_z \geq \frac{\rho}{\underset{n}{max}\,||x_n||} = \frac{\rho}{R}$
$\Rightarrow (\rho_z)^2 \geq (\frac{\rho}{R})^2$
$\Rightarrow (\frac{1}{\rho_z})^2 = U \leq (\frac{R}{\rho})^2 = U_{orig}$
so the answer is $[b]U \leq U_{orig}$
5. $[c]$
We use $w_0 = [0,0,0]$ as the initial vector of PAM and $w_{pla0} = [0,0,0]$ as PLA's
$sign(w_0^\intercal x_1) = +1 \neq y_1 = -1$, update
$w_1 = w_0 + x_1 y_1 = [-1, 2, -2]$
and w_{pla0} also need update
$w_{pla1} = w_{pla0} + x_1 y_1 = [-1, 2, -2]$
$sign(w_1^\intercal x_2) = sign(w_{pla1}^\intercal x_2) = -1 = y_2$,but $y_2 w_1^\intercal x_2 = 7 > \tau$, no need to update
$w_2 = w_1, w_{pla2} = w_{pla1}$
$sign(w_2^\intercal x_3) = sign(w_{pla2}^\intercal x_3) = +1 = y_3$, but $y_3 w_2^\intercal x_3 = 3 < \tau = 5$
PAM needs an update, PLA doesn't need
$w_3 = w_2 + x_3 y_3 = [0, 4, -2]$,
$w_{pla3} = w_{pla2} = [-1, 2, -2]$
$sign(w_3^\intercal x_4) = -1 = y_4$, but $y_4 w_3^\intercal x_4 = 4 < \tau = 5$
$sign(w_{pla3}^\intercal x_4) = -1 = y_4$
PAM needs an update, PLA doesn't need
$w_4 = w_3 + x_4 y_4 = [-1, 5, -2]$,

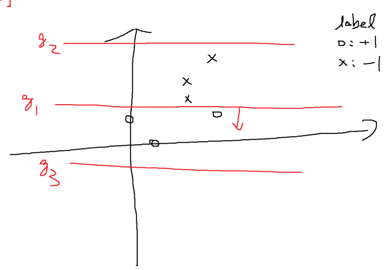
$w_{pla4} = w_{pla3} = [-1, 2, -2]$
$sign(w_4^\intercal x_5) = +1 = y_5$, but $y_5 w_4^\intercal x_5 = 2 < \tau = 5$
$sign(w_{pla4}^\intercal x_5) = -1 \neq y_5$
PAM and PLA need an update
$w_5 = w_4 + x_5 y_5 = [0, 6, -1]$,
$w_{pla5} = w_{pla4} + x_5 y_5 = [0, 3, -1]$
first, we use PAM to predict test samples.
suppose the symbol of test samples are $x_{t1}, \ldots, x_{t4}$ and label $y_{t1}, \ldots, y_{t4}$
$sign(w_5^\intercal x_{t1}) = +1 = y_{t1}$ correct
$sign(w_5^\intercal x_{t2}) = +1 = y_{t2}$ correct
$sign(w_5^\intercal x_{t3}) = +1 = y_{t3}$ correct
$sign(w_5^\intercal x_{t4}) = -1 = y_{t4}$ correct
then we use PLA to predict
$sign(w_{pla5}^\intercal x_{t1}) = -1 \neq y_{t1}$ wrong
$sign(w_{pla5}^\intercal x_{t2}) = -1 \neq y_{t2}$ wrong
$sign(w_{pla5}^\intercal x_{t3}) = +1 = y_{t3}$ correct
$sign(w_{pla5}^\intercal x_{t4}) = -1 = y_{t4}$ correct
PAM predicts all the test samples correctly, and PLA has 2 wrong predictions.
the answers is $[c]2$

2 The Learning Problems

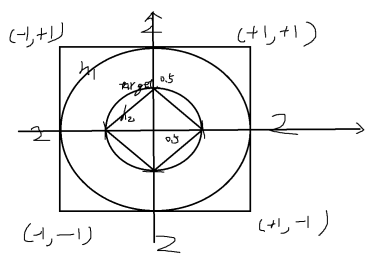
6. (a)
The best fit will be using the regression model because we want to predict a viewer's rate which is a real number between [1,5]. So using a regression model will return a real number that can reflect the customers' rating.
7. $[b]$
I think this would be a binary classification problem. We only have two output values, so we can transform the output to -1 and +1. The labeler's work is to pick the better one, which is similar to the binary classification. We can always assign 1 to the better one and -1 to the worse one by using binary classification.

3 Feasibility of Learning

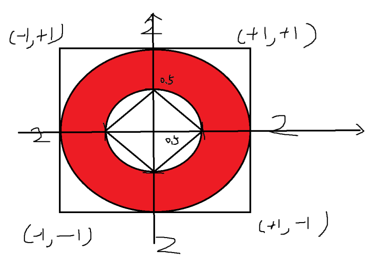
8. [e]



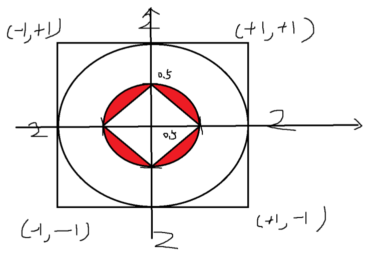
In this question, the hypothesis we care is g_1, g_2, and g_3 in the figure. For g1, we can use (0,2)(label +1), (3,2)(label +1), and (2,3)(label -1) to obtain. When $E_{in}(g_1) = 0$, we have a hyperplane to separate samples perfectly. The $E_{ots}(g_1) = 0$.
This time we look at g_3, and pick (1,0)(label +1), (3,2)(label +1) and (0,2)(label +1) to find hypothesis g_3. g_3 will predict every sample in the figure as label +1, which will lead to $E_{ots}(g3) = 3/3 = 1$.
So now we have the min and max E_{ots}, the answer will be $[e](0, 1)$
9. [d]
In this question, we can draw the target function and two hypotheses on a plane within $[+1, -1] \times [+1, -1]$



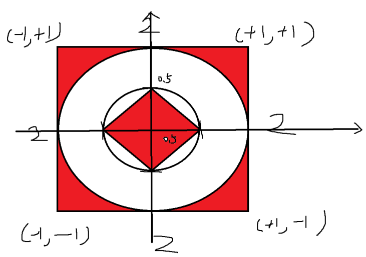
The $E_{out}(h_1)$ will be (red area):



the area is $\pi - \frac{1}{4}\pi = \frac{3\pi}{4}$
and we need to divide it by 4 because the $[+1, -1] \times [+1, -1]$ plane has an area of 4 units.
We now have $E_{out}(h_1) = \frac{3\pi}{4} * \frac{1}{4} = \frac{3\pi}{16}$
The $E_{out}(h_2)$ will be (red area):



the area is $\frac{1}{4}\pi - 0.5$
and we divide it by 4.
We now have $E_{out}(h_2) = (\frac{1}{4}\pi - 0.5) * \frac{1}{4} = \frac{\pi-2}{16}$
The answer of P9 is $[d](E_{out}(h_1), E_{out}(h_2)) = (\frac{3\pi}{16}, \frac{\pi-2}{16})$
10. $[b]$
for h_1 and h_2, if $E_{in}(h_1) = 0 = E_{in}(h_2)$, there are only four parts that samples can locate like the below figure.

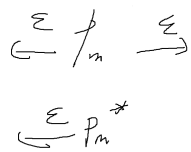


the area of these four parts is $0.5 + (4 - \pi)$, then we divide it by 4, we have $\frac{0.5+(4-\pi)}{4}$
We need to select 4 samples, so the probability will be $(\frac{0.5+(4-\pi)}{4})^4 = 0.01330087221$
The answer is $[b]0.01$
11. $[a]$
Assume
x:number of darts landed in the circle
N: the total darts number
ϵ: 10^{-2}
$P(|4\frac{x}{N} - \pi| > 10^{-2}) \leq 2exp(-2(10^{-2})^2 N)$
We want the probability to be more than 0.99 so we can set $P(|4\frac{x}{N} - \pi| > 10^{-2}) = 0.01$, this step can make sure the estimation error is within 10^{-2} with probability more than 0.99.

$0.01 \geq 2exp(-210^{-2}N)$
$takelnbothside$
$\Rightarrow ln(0.01) - ln(2) \geq -2 \cdot 10^{-4}N$
$\Rightarrow N \geq ln(\frac{2}{0.01}) \cdot \frac{1}{2} \cdot 10^4 \approx 26491\ldots.$

N at least $[a]26492$

12. $[c]$



Since P_m^* is the largest probability, unlike other cases, we only care about one side of the inequality. So the equation we used is below:
$P((P_m^* - P_m) > \epsilon) \leq exp(-2\epsilon^2 N)$
Suppose we have M bins now, we can write down the equation like below:
$P((P_m^* - P_{m_1}) > \epsilon) + P((P_m^* - P_{m_2}) > \epsilon) + \ldots + P((P_m^* - P_{m_M}) > \epsilon) \leq Mexp(-2\epsilon^2 N)$
We need to make sure an ϵ-optimal box with a probability at least $1 - \delta$ if N is large enough.
So we will have the following relation to ensure the bound:
$\delta \geq Mexp(-2\epsilon^2 N)$
take ln
$ln(\frac{\delta}{M}) \geq -2\epsilon^2 N$
$\Rightarrow N \geq ln(\frac{M}{\delta}) \cdot \frac{1}{2}\frac{1}{\epsilon^2}$
Which is the answer $[c]\frac{1}{2\epsilon^2}ln\frac{M}{\delta}$

```python
import numpy as np
from dataclasses import dataclass, field
from typing import List

@dataclass
class PLA:
    M: int # PLA stop after check M randomly-picked sample correct consecutively
    n_init: int
    x_0: float
    scale: float

    def fit(self, x, y):
        """
        Train PLA by x and y
        Args:
            x (List(float) or ndarray): The input training vector space of samples
            y (List(int) or ndarray): The label of input vector space of samples

        Returns:
            PLA class
        """
        # add x_0 to first column
        self.x = np.insert(x, 0, self.x_0, axis=1)*self.scale
        self.y = y

        # the number of training samples
        self.n = len(x)

        # answers
        self.Ein = []
        self.updates = []
        self.w_pla = []
        self.x_0_w_pla = []

        # iteration
        for i in range(self.n_init):
            result = self.run_PLA(self.M)
            # collect answers from different iteration
            self.Ein += [result[0]]
            self.updates += [result[1]]
            self.w_pla += [result[2]]
            self.x_0_w_pla += [result[3]]

        # turn answers list into numpy array
        self.Ein = np.array(self.Ein)
        self.updates = np.array(self.updates)
        self.w_pla = np.array(self.w_pla)
        self.x_0_w_pla = np.array(self.x_0_w_pla)
        return self

    def run_PLA(self, M):
        """
        Run PLA one iter with M random sample correct consecutively as termination condition
        Args:
            M (int): a PLA termination requirement, need to correct M(with random sample) consecutively
times to terminate PLA

        Returns:
            float, int, list(float): the answers of HW
        """
        # if cnt == M, PLA terminate
        cnt = 0
        # count total iteration
        iter = 0
        # init w_t and updates
        w_t = np.zeros(self.x.shape[1])
        updates = 0.0

        while(cnt != M):
            # pick sample randomly, generate one index randomly befor loop
            sample_idx = np.random.randint(self.n, size=1)
            x = self.x[sample_idx].flatten()
            y = self.y[sample_idx].flatten()

            iter += 1
            cnt += 1
            if self.sign(np.dot(w_t, x)) != y:
                updates += 1
                cnt = 0
                w_t += x * y

        # init error and calculate Ein
        error = 0.0
        for i in range(self.n):
            x = self.x[i].flatten()
            y = self.y[i].flatten()
            if self.sign(np.dot(w_t, x)) != y:
                error += 1
        Ein = error/self.n
        return Ein, int(updates), w_t, self.x_0 * w_t[0]

    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 PLA import PLA
```

In [ ]:
```
"""
load data and preprocess
"""
# read data
with open('hw1_train.dat', 'rb') as f:
    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(data[:,0:10])
y = np.reshape(np.array(list(map(int, data[:,10]))), (len(x), 1))
print(x.shape)
print(y.shape)

N = len(x)
```

```
(256, 10)
(256, 1)
```

In [ ]:
```
"""
p13
"""
p13_kwargs = {
    'M': N/2.0,
    'n_init': 1000,
    'x_0': 1.0,
    'scale': 1.0
}

pla_p13 = PLA(**p13_kwargs)
pla_p13 = pla_p13.fit(x, y)
Ein_p13 = pla_p13.Ein
p13_ans = np.mean(Ein_p13)
p13_ans
```

Out[ ]:
```
0.01989453125
```

In [ ]:
```
"""
p14
"""
p14_kwargs = {
    'M': 4.0 * N,
    'n_init': 1000,
    'x_0': 1.0,
    'scale': 1.0
}

pla_p14 = PLA(**p14_kwargs)
pla_p14 = pla_p14.fit(x, y)
Ein_p14 = pla_p14.Ein
p14_ans = np.mean(Ein_p14)
p14_ans
```

Out[ ]:
```
0.00019140625
```

In [ ]:
```
"""
p15
"""
p15_kwargs = {
    'M': 4.0 * N,
    'n_init': 1000,
    'x_0': 1.0,
    'scale': 1.0
}

pla_p15 = PLA(**p15_kwargs)
pla_p15 = pla_p15.fit(x, y)
updates_p15 = pla_p15.updates
p15_ans = np.median(updates_p15)
p15_ans
```

Out[ ]:
```
446.5
```

In [ ]:
```
"""
p16
"""
p16_kwargs = {
    'M': 4.0 * N,
    'n_init': 1000,
    'x_0': 1.0,
    'scale': 1.0
}

pla_p16 = PLA(**p16_kwargs)
pla_p16 = pla_p16.fit(x, y)
wpla_p16 = pla_p16.w_pla
# take all the w0 from wpla and pick median
p16_ans = np.median(wpla_p16[:,0])
p16_ans
```

Out[ ]:
```
34.0
```

In [ ]:
```
"""
p17
"""
p17_kwargs = {
    'M': 4.0 * N,
    'n_init': 1000,
    'x_0': 1.0,
    'scale': 0.5
}

pla_p17 = PLA(**p17_kwargs)
pla_p17 = pla_p17.fit(x, y)
updates_p17 = pla_p17.updates
p17_ans = np.median(updates_p17)
p17_ans
```

Out[ ]:
```
444.0
```

In [ ]:
```
"""
p18
"""
```

```
p18_kwargs = {
    'M': 4.0 * N,
    'n_init': 1000,
    'x_0': 0,
    'scale': 1.0
}

pla_p18 = PLA(**p18_kwargs)
pla_p18 = pla_p18.fit(x, y)
updates_p18 = pla_p18.updates
p18_ans = np.median(updates_p18)
p18_ans
```

Out[ ]:
```
448.0
```

In [ ]:
```
"""
p19
"""
p19_kwargs = {
    'M': 4.0 * N,
    'n_init': 1000,
    'x_0': -1.0,
    'scale': 1.0
}

pla_p19 = PLA(**p19_kwargs)
pla_p19 = pla_p19.fit(x, y)
x_0_w_pla_p19 = pla_p19.x_0_w_pla
p19_ans = np.median(x_0_w_pla_p19)
p19_ans
```

Out[ ]:
```
34.0
```

In [ ]:
```
"""
p20
"""
p20_kwargs = {
    'M': 4.0 * N,
    'n_init': 1000,
    'x_0': 0.1126,
    'scale': 1.0
}

pla_p20 = PLA(**p20_kwargs)
pla_p20 = pla_p20.fit(x, y)
x_0_w_pla_p20 = pla_p20.x_0_w_pla
p20_ans = np.median(x_0_w_pla_p20)
p20_ans
```

Out[ ]:
```
0.44375660000000006
```