HTML

Homework 1

B07502166 魏子翔

1. [a] is best suited for machine learning because the data can be easily collected, and it's almost impossible to create a precise definition of the function.
   There's no underlying pattern in [b].
   There's a many rule-based algorithm for [c] such as BFS, DFS,... and so on.
   There's no one knows how Zeus actually looks like. So it's hard to collect the data.

2. If $\mathbf{w}_{t+1}^T$ is correct on $(\mathbf{x}_{n(t)}, y_{n(t)})$, then $y_{n(t)}\mathbf{w}_{t+1}^T\mathbf{x}_{n(t)} > 0$

$$\begin{aligned}
y_{n(t)}\mathbf{w}_{t+1}^T\mathbf{x}_{n(t)} &= y_{n(t)}\left(\mathbf{w}_t + y_{n(t)}\mathbf{x}_{n(t)} \cdot \left\lfloor \frac{-y_{n(t)}\mathbf{w}_t^T\mathbf{x}_{n(t)}}{||\mathbf{x}_{n(t)}||^2} + 1 \right\rfloor\right)^T \mathbf{x}_{n(t)} \\
&= y_{n(t)}\mathbf{w}_t^T\mathbf{x}_{n(t)} + \mathbf{x}_{n(t)}^T \cdot \left\lfloor \frac{-y_{n(t)}\mathbf{w}_t^T\mathbf{x}_{n(t)}}{||\mathbf{x}_{n(t)}||^2} + 1 \right\rfloor \mathbf{x}_{n(t)} \\
&= y_{n(t)}\mathbf{w}_t^T\mathbf{x}_{n(t)} + \left\lfloor \frac{-y_{n(t)}\mathbf{w}_t^T\mathbf{x}_{n(t)}}{||\mathbf{x}_{n(t)}||^2} + 1 \right\rfloor ||\mathbf{x}_{n(t)}||^2 \\
&= y_{n(t)}\mathbf{w}_t^T\mathbf{x}_{n(t)} + (-y_{n(t)}\mathbf{w}_t^T\mathbf{x}_{n(t)}) + ||\mathbf{x}_{n(t)}||^2 \\
&= ||\mathbf{x}_{n(t)}||^2 > 0
\end{aligned}$$

   So, [d] ensures that $\mathbf{w}_{t+1}^T$ is correct on $(\mathbf{x}_{n(t)}, y_{n(t)})$

3. In the original definition, $R^2 = \max_n ||\mathbf{x}_{n(t)}||^2$. With the normalized inputs, R^2 is always equal to 1 because of the normalization results. So, the upper bound is $\frac{1}{\rho_z^2}$, which is [c].

4. $\rho_z = \min_n \frac{y_n\mathbf{w}_f^T\mathbf{z}_n}{||\mathbf{w}_f||} = \min_n \frac{y_n\mathbf{w}_f^T\mathbf{x}_n}{||\mathbf{w}_f||} \min_n \frac{1}{||\mathbf{x}_n||} = \min_n \frac{y_n\mathbf{w}_f^T\mathbf{x}_n}{||\mathbf{w}_f||} \frac{1}{\max_n ||\mathbf{x}_n||} = \frac{\rho}{R}$
   So, the new bound $U = \frac{1}{\rho_z^2} = \left(\frac{R}{\rho}\right)^2$, which is equal to the original bound U_{orig}, [c].

5. PLA:

training:

i	$\mathbf{w}_i$	$\mathbf{x}_i$	y_i	$\mathbf{w}_i^T\mathbf{x}_i$	update?	$\mathbf{w}_{i+1}$
0	(0, 0, 0)	(1, -2, 2)	-1	0	yes	(-1, 2, -2)
1	(-1, 2, -2)	(1, -1, 2)	-1	-7	no	(-1, 2, -2)
2	(-1, 2, -2)	(1, 2, 0)	1	3	no	(-1, 2, -2)
3	(-1, 2, -2)	(1, -1, 0)	-1	-3	no	(-1, 2, -2)
4	(-1, 2, -2)	(1, 1, 1)	1	-1	yes	(0, 3, -1)

testing:

i	$\mathbf{w}_i$	$\mathbf{x}_i$	y_i	$\mathbf{w}_i^T\mathbf{x}_i$	correct?
0	(0, 3, -1)	(1, 0.5, 2)	1	-0.5	no
1	(0, 3, -1)	(1, 0.25, 1)	1	-0.25	no
2	(0, 3, -1)	(1, 0.5, 0)	1	0.5	yes
3	(0, 3, -1)	(1, -0.5, 1)	-1	-2.5	yes

PAM:

training:

i	$\mathbf{w}_i$	$\mathbf{x}_i$	y_i	$y_i\mathbf{w}_i^T\mathbf{x}_i$	update?	$\mathbf{w}_{i+1}$
0	(0, 0, 0)	(1, -2, 2)	-1	0	yes	(-1, 2, -2)
1	(-1, 2, -2)	(1, -1, 2)	-1	7	no	(-1, 2, -2)
2	(-1, 2, -2)	(1, 2, 0)	1	3	yes	(0, 4, -2)
3	(0, 4, -2)	(1, -1, 0)	-1	4	yes	(-1, 5, -2)
4	(-1, 5, -2)	(1, 1, 1)	1	3	yes	(0, 6, -1)

testing:

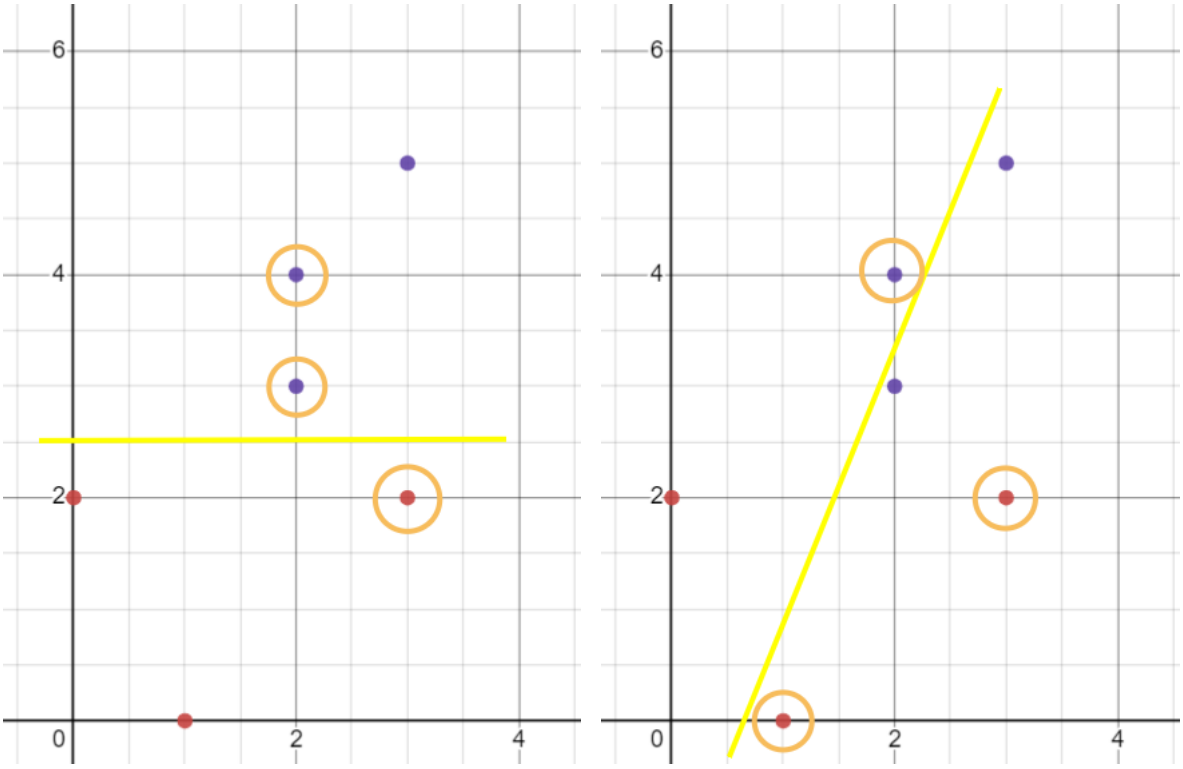
i	$\mathbf{w}_i$	$\mathbf{x}_i$	y_i	$\mathbf{w}_i^T\mathbf{x}_i$	correct?
0	(0, 6, -1)	(1, 0.5, 2)	1	1	yes
1	(0, 6, -1)	(1, 0.25, 1)	1	0.5	yes
2	(0, 6, -1)	(1, 0.5, 0)	1	3	yes
3	(0, 6, -1)	(1, -0.5, 1)	-1	-4	yes

So there's 2 examples are wrongly predicted by PLA but correctly predicted by PAM.

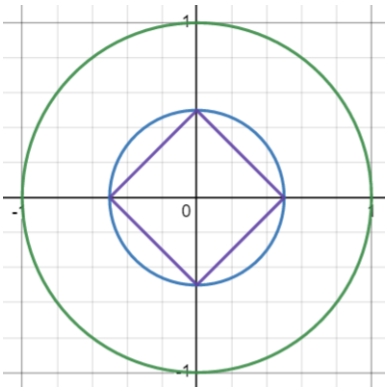
6. The training data should contain all y_n, and the outputs are real number within [1, 5], so it should be supervised regression.

7. We can define one of the two inputs to be +1, and another to be -1, then we can use binary classification to decide which one of the two task is better by output +1 or -1.

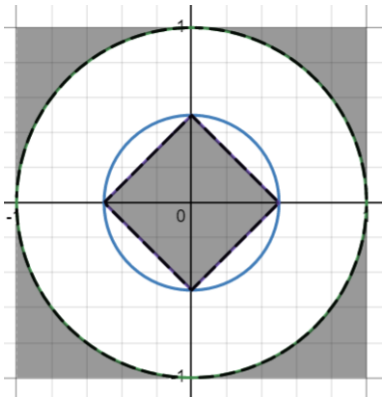
8. Below are two hypothesis after some magical optimizations, $y = +1$ for those red points, while $y =$ -1 for those purple points, and the circles represent the points that we chose as the training set, the yellow line is the final hypothesis. For the left one, we can say that it's a perfect hypothesis because that we can do correct prediction on the all testing set. However, for the right one, we fail on the all testing set. So the answer is (0, 1).



9. We can get the answer by calculating the area difference between the target function and the two hypotheses, as the figure below. The small circle is the target function, and the bigger circle is h_1, while the diamond is h_2.
So, $E_out(h_1) = \frac{\pi - 0.25\pi}{4} = \frac{3\pi}{16}$, $E_out(h_1) = \frac{0.25\pi - 0.5}{4} = \frac{\pi - 2}{16}$, which is [d].



10. This problem is same as finding the area that inside the all three function or outside the all three function, which is the gray area in the below figure. So, the probability that 4 examples $E_{in}(h2) = E_{in}(h1) = 0$ is $\left(\frac{4-\pi+0.5}{4}\right)^4 = 0.3396^4 = 0.013$, which is [b].



11. $0.01 = P(|4p - \pi| > 0.01) = P(|p - \frac{\pi}{4}| > 0.0025) \leq 2\exp(-2 \cdot 0.0025^2 \cdot N)$
    $\Rightarrow \ N = \frac{\ln(0.01/2)}{-2 \cdot 0.0025^2} = 423865.3$
    $\Rightarrow$ N=432866 is the smallest value.

12. Consider one-sided Hoeffding inequality, $P(\nu - \mu > \epsilon) \leq \exp(-2\epsilon^2 N)$
    $\delta \leq M\exp(-2\epsilon^2 N)$
    $\Rightarrow \ \ln\frac{\delta}{M} \leq -2\epsilon^2 N$
    $\Rightarrow \ N \geq \frac{1}{2\epsilon^2}\ln\frac{M}{\delta}$
    $\Rightarrow$ [c]

```
13 import numpy as np

   def read_data(filename):
       X, y = [], []
       with open(filename, "r") as f:
           for line in f:
               line = line[:-1]
               line = list(map(float, line.split()))
               X.append(line[:-1])
               y.append(line[-1])
       return np.array(X), np.array(y)

   def sign(c):
       if np.sign(c) == -1:
           return -1
       return 1

   def solver(X, y, problem):
       N = 256
       M = N//2 if problem == 13 else N*4
       update_list = []
       err_list = []
       w0_list = []
       correct = 0
       if problem in [13, 14, 15, 16, 17]:
           X_new = np.concatenate((np.ones(N)[:, None], X), axis=1)
       elif problem == 18:
           X_new = np.concatenate((np.zeros(N)[:, None], X), axis=1)
       elif problem == 19:
           X_new = np.concatenate((np.ones(N)[:, None]*(-1), X), axis=1)
       elif problem == 20:
           X_new = np.concatenate((np.ones(N)[:, None]*0.1126, X), axis=1)
       if problem == 17:
           X_new = X_new/2
       for i in range(1000):
           np.random.seed(i*5)
           W = np.zeros(11)
           update = 0
           while True:
               idx = np.random.randint(0, N)
               h = np.dot(W, X_new[idx])
               if sign(h) != sign(y[idx]):
                   W += y[idx]*X_new[idx]
                   correct = 0
                   update += 1
```

```
                continue
            correct += 1
            if correct == M:
                break
        update_list.append(update)
        w0_list.append(W[0]*X_new[0][0])
        err = 0
        for i in range(len(X_new)):
            h = np.dot(W, X_new[i])
            if sign(h) != sign(y[i]):
                err += 1
        err_list.append(err/N)

    if problem in [13, 14]:
        print(np.average(err_list))
    elif problem in [15, 17, 18]:
        print(np.median(update_list))
    elif problem in [16, 19, 20]:
        print(np.median(w0_list))

if __name__ == "__main__":
    filename = "hw1_train.dat"
    X, y = read_data(filename)

    #to run a single problem
    solver(X, y, 13)

    #to run all problems
    for pro in range(13, 21):
        solver(X, y, pro)
```