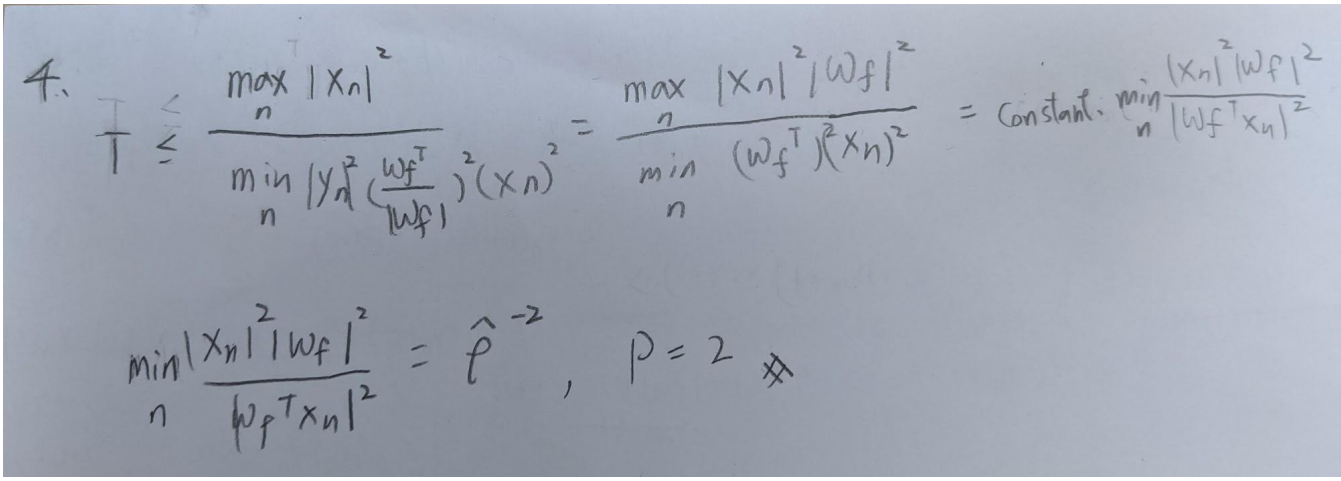
Homework #1
Machine Learning Foundations (NTU, Fall 2020)

b05901060
賴繹文

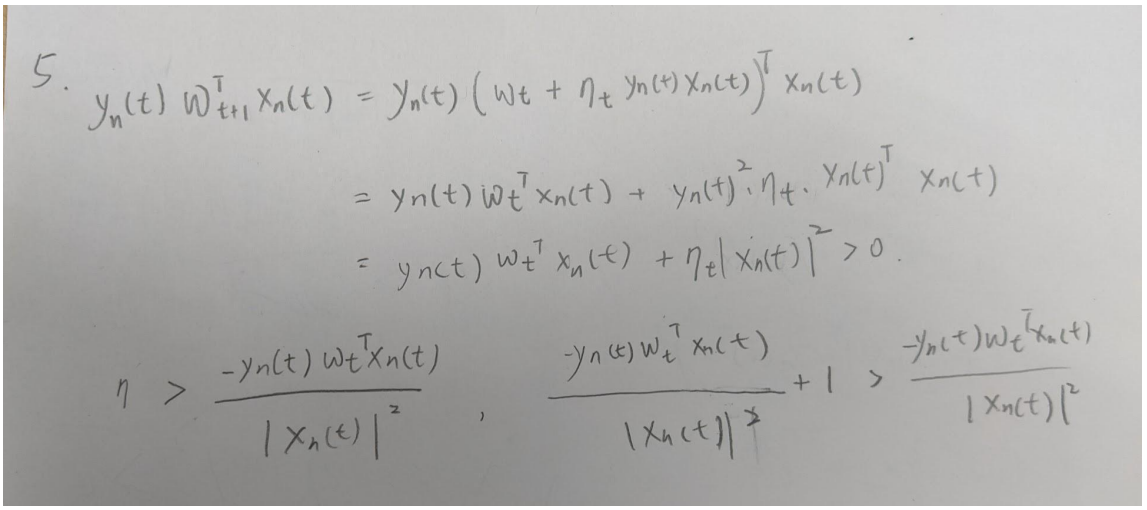
1. **d**.
   The quality of mangoes can be simply discovered by its appearance, which is a form in pictures. By telling the program which kind of images close to the high or low quality, we can train a model to rank those mangoes.
2.
   a. No. This is just a result of a probability, containing nothing about ML.
   b. No. This is also another result of probability, which is more unclear and can't be described as Math, compared with (a)
   c. No. The list of words may not correctly contain all spam words, and calculating numbers of appearance in mail just a simple work, not ML.
   d. No. Though there are some rules about how to discern the spam mail, it just follow the rules and generate results. There's nothing to be optimized and learned by program, thus it's not ML.
   e. **Yes**. There is a goal, spam score, for systems to learn and optimize, and decide a binary result for each mail.
3. **d**.

   Unchanged. Since $T <= R^2 \rho^2 = max |x_n|^2 / (y_n^2 (x_f^2 / |x_f|^2)^2 |x_n|^2)$

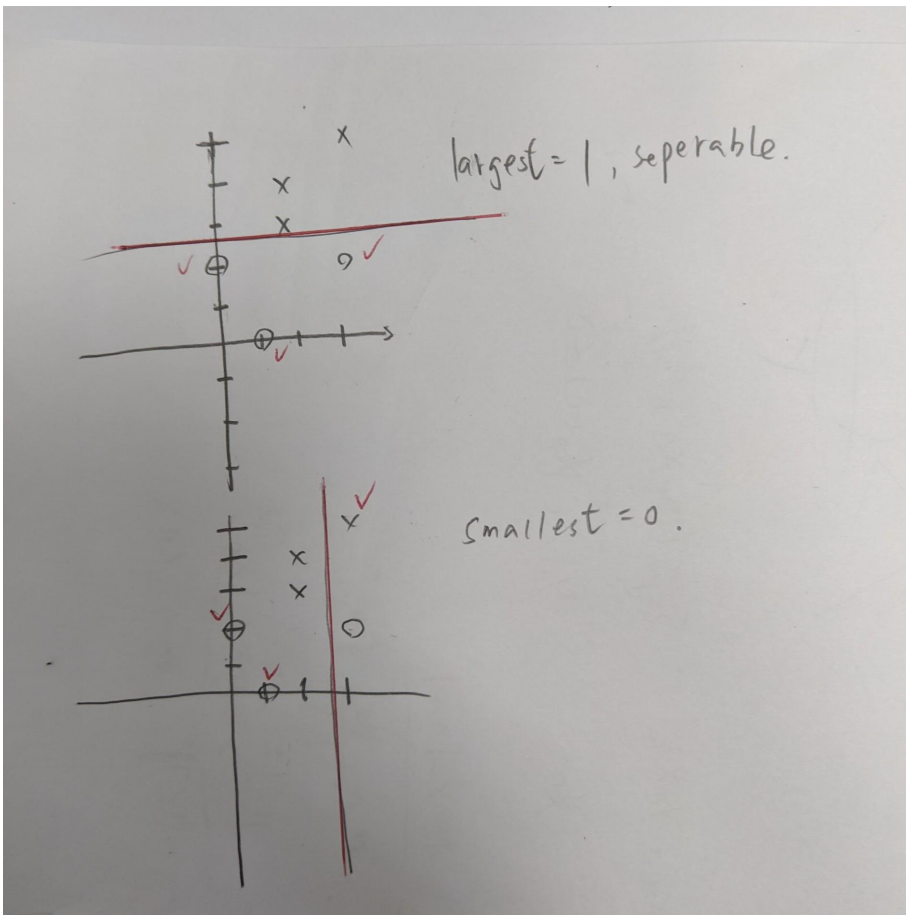
   the constant after the scale down is $(1/4)^2/(1/4)^2 = 1$, thus the speed, or the mistakes needed to be corrected don't change.
4. **c.**



5. **d.**

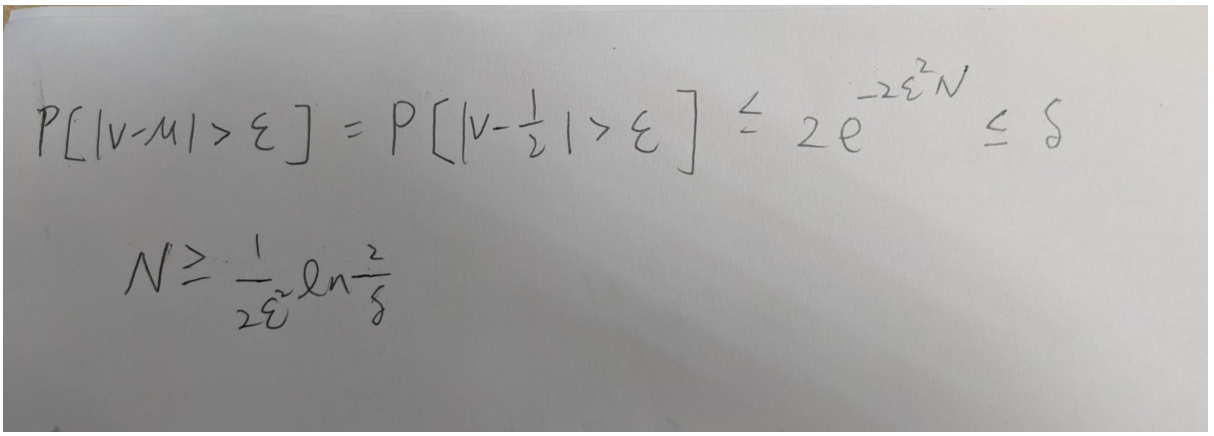


6. **a, b, c, d**
   Since the correcting directions of a,b,c,d are the same, the differences are the speed of termination. Thus, they will all get perfect lines, but the *e* with wrong direction won't.
7. **e**
   There are no clear rules about how AI should learn, and there is only implicit sign from the result of the games. No labels and categories on the inputs, so it won't be semi-supervised or supervised. No human includes in training indicate that it's not human learning. The goal isn't clustering, which is most suitable for unsupervised one. Hence, it is reinforcement learning.
8. **b.**
   Structured learning: There are no explicit classes for self-driving learning.
   Semi-supervised learning: There are videos which are how humans act and without human records.
   Batch learning: Bunch of datas go through input side once, not be fed in sequentially.
   Raw feature: The inputs of video couldn't be represented by vectors or sophisticated physical meaning.
9. **e**

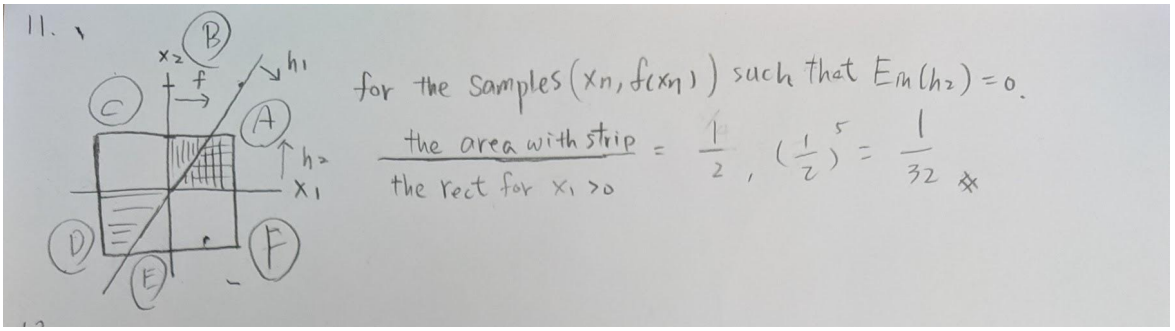


10. **b**

The more probable side with prob. at least 1 - delta is equal to that the “one side” is with prob. less then delta.

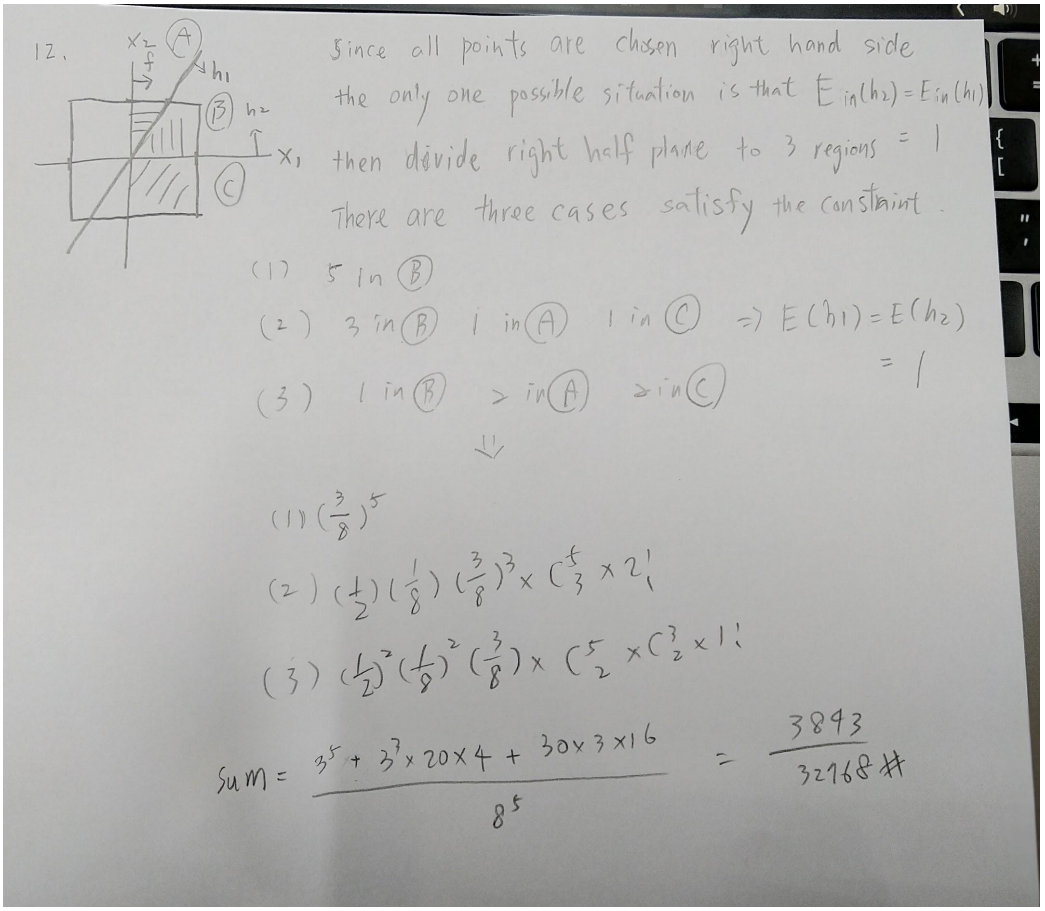


11. c



Here I choose the wrong area for E(h2) = 0. It should be the square with x1 > 0 & x2 < 0. While the answer is the same.

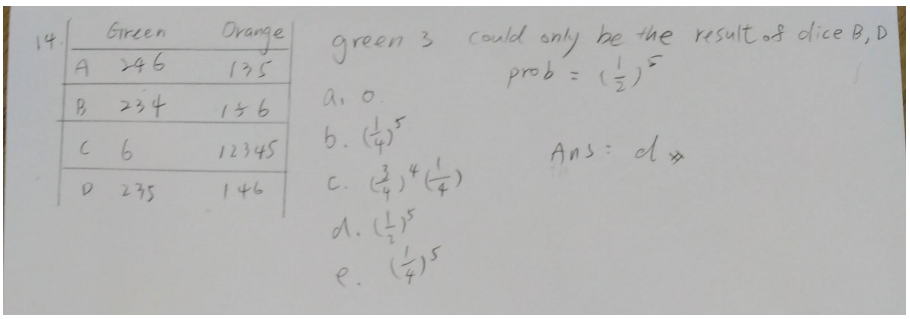
12. d



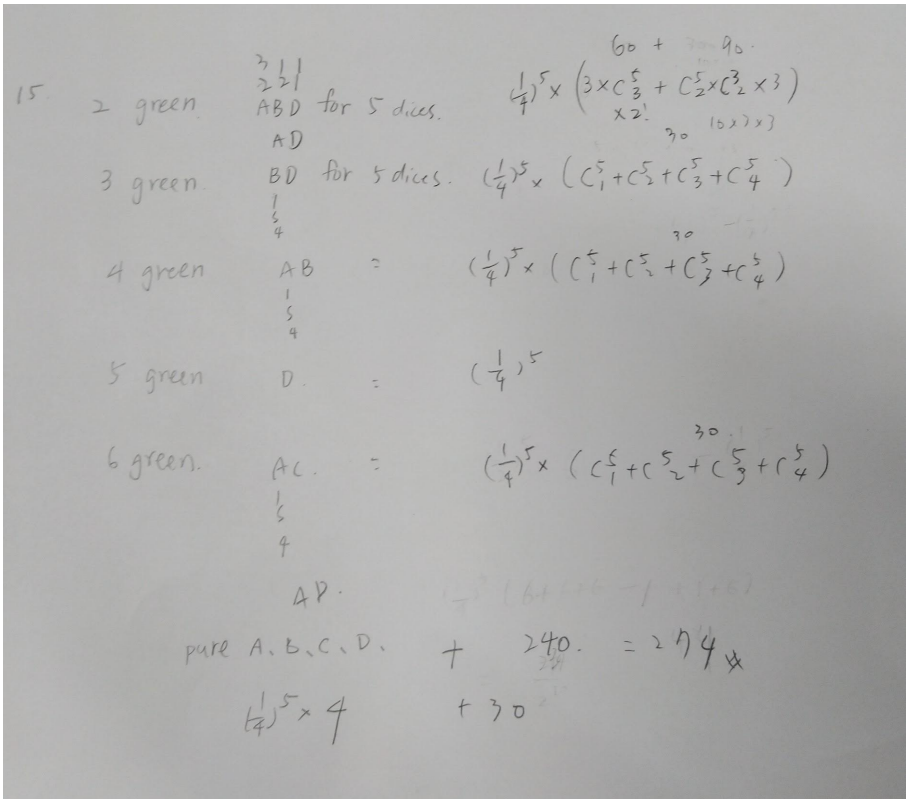
13. b

Since the bad data for hi(x), i = 1 ~ d, is the same as the bad data for hi(x), i = d+1 ~ 2d. We need to only calculate the bound for hi(x), i = 1 ~ d.
There are d hypotheses set, thus C = d

14. c



15. c



16. b, 11
17. b, -7
18. c, 15
19. d, 17
20. d, 17

Source Code:

```
import numpy as np
import random
import statistics
data = []
with open('hw1_dat', 'r') as f:
   d = f.readlines()
   for i in d:
```

```
        k = i.rstrip().split("  ")
        data.append([float(i) for i in k])
data = np.array(data, dtype='O')
update_list = []
w_zero = []

for i in range(999):
    random.seed(i)
    w = np.zeros(11)
    n_update = 0
    n_counter = 0
    while(True):
        n_r =  random.randint(0,99)
        sign = 0
        sums = 0.0
        for index in range(11): # sum of w_i * x_i,  x_0 = 1
            if index == 10:
                sums += w[index] * 1 #x_0
            else:
                sums += w[index] * (data[n_r][index])
        #print(sums)
        if sums <= 0.0:
            sign = -1.0
        else:
            sign = 1.0

        if (sign) != float(data[n_r][10]):
            n_counter = 0                    # need consecutive
500 times
            n_update += 1
            for index in range(11): #  W_t+1 = W_t + y_n * x_n
                if index == 10:
```

```
                    w[index] += 1 * float(data[n_r][10]) #x_0
                else:
                    w[index] += (data[n_r][index]) *
(data[n_r][10])
            #print("update",n_update,n_r,i)
        else:
            n_counter += 1
            #print("count",n_counter,i)
        if n_counter == 500:
            #print(i,n_update)
            break
    update_list.append(n_update)
    w_zero.append(w[10])
print(statistics.median(update_list))
print(statistics.median(w_zero))
```