CS/CNS/EE 156a

Homework 1

Sung Hoon Choi

1. Answer: [d]

(i): Not Learning.

Since we know the exact coin specifications, we already know the target function.

(ii): Supervised Learning

Since we have a set of labeled data(coins), it is a supervised learning.

(iii): Reinforcement Learning

Although we don't have any labelled data, we have the reward/penalty for every move to be used for optimizing our strategy. Thus, it is a reinforcement learning.

2. Answer: [a]

(i) We can pin down the principle of prime number mathematically. Thus, it's not suited for machine learning

(ii) Definitely there is a pattern of frauds. (ex. Speak loud,, smile often, etc..) However, we cannot pin down the pattern of frauds mathematically. Also, we would have the information of past frauds. Thus, it's suited for machine learning.

(iii) The time of object falling to ground can be derived mathematically by using physics equations. Thus, it's not suited for machine learning.

(iv) There exists a pattern that describes the best cycle of traffic lights that perform well in a busy intersection. However, it is hard to calculate the exact mathematical equation of the traffic lights cycle. Besides, we would have the data on each traffic light cycle and its effectivity in a busy intersection. Thus, it is well suited for machine learning.

3. Answer: [d]

i) $\mathrm{P(the\ first\ ball\ is\ black)}$

$= \mathrm{P(picking\ first\ bin)} * \mathrm{P(picking\ black\ from\ first\ bin)} + \mathrm{P(picking\ second\ bin)} * \mathrm{P(picking\ black\ from\ second\ bin)}$

$= \frac{1}{2} * 1 + \frac{1}{2} * \frac{1}{2} = \frac{3}{4}$

ii) $P(\text{the first ball is black } \cap \text{ the second ball is black})$

$= \frac{1}{2}$

$\because$ The probability that both first ball and second ball are black is simply $\frac{1}{2}$ because this can happen only if you pick the first bin, which has 2 black balls.

Therefore, by i) and ii),

$\mathrm{P(the\ second\ ball\ is\ black\ |\ the\ first\ ball\ is\ black)}$

$= \frac{P(\text{the first ball is black } \cap \text{ the second ball is black})}{P(\text{the first ball is black})}$

$= \frac{1/2}{3/4}$

$= \frac{2}{3}$

4. Answer: [b]

$\mathrm{P}(\nu = 0)$

$= (1 - \mathrm{P(Drawn\ ball\ is\ red)})^{10} = (1 - 0.55)^{10} = 0.000340506289 \approx 3.405 * 10^{-4}$

5. Answer: [c]

By Problem 4,

$\mathrm{P}(\nu \neq 0) = 1 - 3.405 * 10^{-4} = 0.9996595$ for each sample.

Therefore, the probability that at least one of the samples has $\nu = 0$ is

1-P(all 1000 samples have $\nu \neq 0$)

$= 1 - \mathrm{P}(\nu \neq 0)^{1000}$

$= 1 - 0.9996595^{1000}$

$= 0.288626722$

6. Answer: [e]

Let's find the full list of possible target functions:

Since the outputs for the given five examples are fixed, we can simply focus on the three remaining points:

$$\boldsymbol{x_n} = 101, 110, 111.$$

Then, the possible target functions are:

<table>
<tr><th>$\boldsymbol{x_n}$</th><th>101</th><th>110</th><th>111</th></tr>
<tr><td rowspan="8">y_n</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>

Thus, if we calculate the scores using this table, we get:

For [a], $(1) * 3 + (3) * 2 + (3) * 1 + (1) * 0 = 3 + 6 + 3 + 0 = 12$

For [b], $(1) * 3 + (3) * 2 + (3) * 1 + (1) * 0 = 3 + 6 + 3 + 0 = 12$

For [c], $(1) * 3 + (3) * 2 + (3) * 1 + (1) * 0 = 3 + 6 + 3 + 0 = 12$

For [d], $(1) * 3 + (3) * 2 + (3) * 1 + (1) * 0 = 3 + 6 + 3 + 0 = 12$

Therefore, the scores for all options are same. Thus, the answer is [e].

7. Answer: [b]

I got the average iterations of 9.806 for N=10. Please see the program code attached below for derivation.

8. Answer: [c]

I got the average error rate of 0.109. I derived this value by generating a large set of test data, not by the mathematical derivation. Please see the program code attached below for derivation.

9. Answer: [b]

I got the average iterations of 94.049. Please see the program code attached below for derivation.

10. Answer: [b]

I got the average error rate of 0.0144. I derived this value by generating a large set of test data, not by the mathematical derivation. Please see the program code attached below for derivation.

Code for Problem 7~10

```
#Sung Hoon Choi
#CS/CNS/EE156a HW1 Problem 7~10

import numpy as np

def gen_target_func():              #generate a target function(f(x)) and return the corresponding vertical coordinate
                                     #input: none
                                     #output: target_function. format: [slope, y_intercept]
    rnd_x1 = np.zeros(2)
    rnd_x2 = np.zeros(2)

    for i in range (0,2):
        rnd_x1[i] = (1 if np.random.rand(1) < 0.5 else -1) * np.random.rand(1)

    for i in range (0,2):
        rnd_x2[i] = (1 if np.random.rand(1) < 0.5 else -1) * np.random.rand(1)

    slope_target_func = (rnd_x2[1]-rnd_x1[1])/(rnd_x2[0]-rnd_x1[0]) # slope = (y2-y1)/(x2-x1)

    y_intercept = rnd_x2[1]-slope_target_func*rnd_x2[0]

    return [slope_target_func, y_intercept]

def Label_data(X_vector, target_f): #return a correct label(1 or -1) by using the input vector and target equation f.
                                     #inputs
                                     # X_vector : input point's coordinate. format: [a, b]
                                     # target_f : target function. format: a
                                     #outputs
                                     # y : correct label for the input vector. format: a (1 or -1)

    if(X_vector[1] > target_f):    #if the input's vertical coordinate is above the target function, return 1 label
                                    #if the input's vertical coordinate is below the target function, return -1 label
        return 1
    else:
        return -1

def generate_random_point():       #generate random data point's coordinate
                                    #inputs
                                    # none
                                    #outputs
                                    # x: random points. format: [a,b]
    x = np.zeros(2)
    x[0] = (1 if np.random.rand(1) < 0.5 else -1) * np.random.rand(1)
    x[1] = (1 if np.random.rand(1) < 0.5 else -1) * np.random.rand(1)
    return x


#Initializing the constants which will be used for training and testing
Total_iteration = 0         #Total iterations over all runs
Total_Run = 1000            #Total number of runs
Test_Data_Num = 1000        #Total number of testing data (used for testing the hypothesis g(x))
Total_Wrong_Points = 0      #Total number of f(x) != g(x) over all runs

#Training begins
for run in range (0,Total_Run):
    N=100
    target_info = gen_target_func()     #target_info = [slope_target_func, y_intercept]

    x = np.zeros([N,4])         # x - [[1, x1,x2,label(y)],
                                #      [1, x1,x2,label(y)],
                                #      ..
                                #      [1, x1,x2,label(y)]]

    w = np.zeros([3,1])         #initializing w vector
    w = np.squeeze(w)           #remove one dimension for matrix operations

    # generate N random data points with their correct labels based on the current target function f(x)
    for i in range (0, N):
        x[i,0] = 1                                  #x0 = 1
        x[i,1:3] = generate_random_point()          #random data points coordinate data
        f_x = target_info[0] * x[i,1] + target_info[1] #obtaining the target equation f
        x[i,3] = Label_data(x[i,1:3],f_x)           #using f, obtain the label(y) and append it to the array
```

```
    iteration = 0
    mismatch = 0
    for i in range (0, 10000):
        random_gen = (int) (np.random.rand(1)*N)  #pick random misclassified points
        g_x = np.dot(w.T, x[random_gen,0:3])      #g(x) = dot(w,x)
        if(x[random_gen,3] != np.sign(g_x)):      #if y is not equal to the sign of g(x)
            w = w + x[random_gen,3]*x[random_gen,0:3]     # w = w + y*X
            iteration = iteration + 1


#Testing begins
    #Generate random points to examine the error rate of g(x)
    test = np.zeros([Test_Data_Num, 4])
    for i in range(0, Test_Data_Num):
        test[i, 0] = 1  # x0 = 1
        test[i, 1:3] = generate_random_point()
        f_x = target_info[0] * test[i, 1] + target_info[1]
        test[i, 3] = Label_data(test[i, 1:3], f_x)  # y (label)

    #Examine the error using the generated test points
    wrong = 0
    for i in range (0,Test_Data_Num):
        g_x = np.dot(w.T, test[i, 0:3])  # g(x) = dot(w,x)
        if (test[i, 3] != np.sign(g_x)):
            wrong = wrong + 1            #if the hypothesis is wrong, increase the counter 'wrong'.

    Total_Wrong_Points = Total_Wrong_Points + wrong     #Add up the number of all wrong points
    Total_iteration = Total_iteration + iteration       #Add up the number of iterations


print('Total iterations: ', Total_iteration)
print('Average iterations: ', Total_iteration/Total_Run)
print('Total Wrong Points: ', Total_Wrong_Points)
print('Error Rate: ', Total_Wrong_Points/(Test_Data_Num*Total_Run))
                                                    #Test_Data_Num is the number of test data for each run
                                                    #So we have to divide the total number of wrong points by
                                                    #(Test_Data_Num*Total_Run) to obtain the accurate error rate.
```