P.S. 1

Tuesday, September 26, 2023 5:21 PM

1) i) No learning is happening here because the machine already knows what the size of the coins are and the statistical frequency with which they appear. It's not uncovering or learning anything about how to recognize the coins. Rather, it's told how to recognize the coins (known target function) and the machine simply follows that algorithm.

ii) This is an example of supervised learning because the machine is handed a dataset of labelled coins with the format (input, correct output). Since the machine has to learn where the boundaries are between clusters based on labelled data, this is supervised learning.

iii) This is an example of reinforcement learning because the computer is working off (input, some output, grade for the output) and adjusting itself by looking at the grade of the output (negative) after some input (losing moves). By doing this repeatedly, the computer is able to learn which inputs lead to outputs with a high grade eventually leading to a winning model.

$\boxed{\text{The correct answer is d}}$

2) Both (i) & (iii) would not be suitable examples of machine learning because the task can be pinned down mathematically with complete accuracy. For example, one way to accomplish

task (i) is by the sieve of Erastothenes. Task (iii) can be accomplished by simple Kinematics equations & Newtonian mechanics.

Task (ii) is a good example of a task suited for machine learning because:

a) a pattern exists → a user will generally charge their credit card in a way that's consistent across factors like transaction amount, transaction location etc. For example, if a user in Florida typically charges the card for $50-100 for each transaction, a purchase of $1500 from Arizona would be suspicious and break the pattern

b) We cannot pin it down mathematically
↳ It's impractical to come up with a mathematical function that models the credit utilization patterns of millions of users each having their own complex factors controlling their patterns. The computational power needed to come up with such an equation would be staggering, and even if one could, who's to say the equation won't have to be readjusted when new users join.

c) We have data on it
↳ Any credit lending company has access to the credit utilization & repayment habits of its users, and multiplied across millions of users, there is more than enough data to train a ML model.

Task (iv) is also a suitable ML task because:

a) a pattern exists → there is a correlation b/w factors like

a) a pattern exists → there is a correlation b/w factors like # of lanes, expected traffic, rush hours, city/rural? and the optimal traffic light cycle

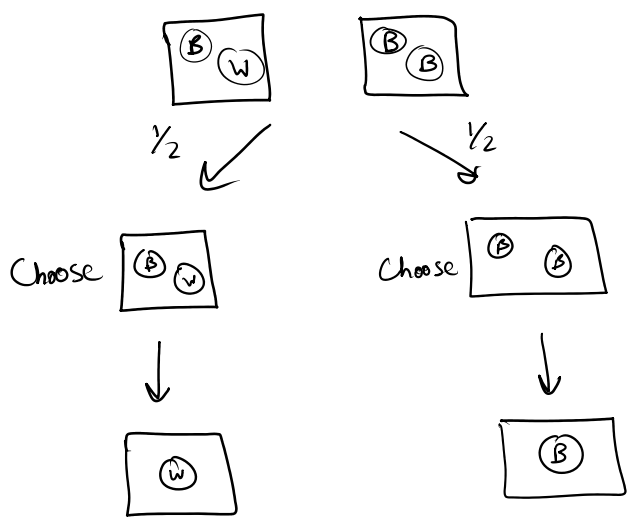
b) we cannot pin it down mathematically

↳ due to the sheer number of factors contributing to the busyness of an intersection, it would be impossible to come up with a single mathematical equation that models all busy intersections

c) we have data on it → the U.S. Department of Transportation's Federal Highway Administration collects traffic data which can be used to train a model. Several local towns/municipalities and states also collect data which can be used.

$\boxed{\text{The answer is a}}$

3)



Because one of the bags is picked randomly at the beginning, we know that there are two outcomes that could happen. Since we know the ball we picked is black, the other ball must be either black or white (since we pick out of the same bag). Whether it's black or white was determined when the bag was chosen, so there is a 50% chance the second ball is also black dependent solely on the first random event.

$\boxed{\text{The answer is c}}$

4) Since we are only looking at one sample, all we have to do is look at the probability of of drawing one green marble ($1-u = 1-0.55 = 0.45$) and raise it to the sample size.

$$P(\nu=0) = (1-0.55)^{10} = 3.405 \times 10^{-4}$$

$\boxed{\text{The answer is b}}$

5) In the previous problem, we calculated the probability of one sample with no reds. The probability of having a sample with at least one red will therefore be 1 - (probability calculated in 4). Repeating this process 1000 times gives us the probability of all the samples having at least one red in the run of 1000

samples. We can then subtract this from 1 to get the probability of at least one sample having no reds.

$P(\nu = 0 \text{ in one sample}) = 3.405 \times 10^{-4}$

$P(\nu > 0 \text{ in one sample}) = 1 - 3.405 \times 10^{-4}$

$P(\nu > 0 \text{ in all the samples in 1000 trials}) = (1 - 3.405 \times 10^{-4})^{1000}$

$P(\nu = 0 \text{ in at least one sample in 1000 trials}) = 1 - (1 - 3.405 \times 10^{-4})^{1000}$

$= 0.28863119785$

$\boxed{\text{The answer is c}}$

6) Since there are only 8 possible target functions, we can just enumerate the output of these 8 functions on the points outside D. After that, we can look at the output of each g for the set of three points and evaluate the score for each g.

x_n	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7
101	0	0	0	1	0	1	1	1
110	0	0	1	0	1	0	1	1
111	0	1	0	0	1	1	0	1
	1	0	2	2	1	1	3	2

all possible target functions have binary outputs

x_n	g_0	g_1	g_2	g_3
101	1	0	0	1
110	1	0	0	1
111	1	0	1	0

Score = (# of target functions agreeing with hypothesis on all 3 points)×3 + (# of target functions agreeing with hypothesis on exactly 2 points)×2 + (# of target functions agreeing with hypothesis on exactly 1 point)×1 + (# of target functions agreeing with hypothesis on 0 points)×0.

$g_0 \rightarrow$ score $= (1)\times 3 + (3)\times 2 + (3)\times 1 + (1)\times 0 = 12$

$g_1 \rightarrow$ score $= (1)\times 3 + (3)\times 2 + (3)\times 1 + (1)\times 0 = 12$

$g_2 \rightarrow$ score $= (1)\times 3 + (3)\times 2 + (3)\times 1 + (1)\times 0 = 12$

$g_3 \rightarrow$ score $= (1)\times 3 + (3)\times 2 + (3)\times 1 + (1)\times 0 = 12$

Since all the hypotheses have the same score in terms of agreement with the possible target functions

$\boxed{\text{the answer is e}}$

For Questions 7-10, I was able to write up a version of the PLA algorithm, but I didn't have enough time to run the algorithm and analyze the results with the varying data sizes. I've attached the code I wrote and hedged my answers for the multiple choice.

7) D & E
8) D & E
9) D & E
10) D & E

In [ ]:
```python
import numpy as np
import matplotlib.pyplot as plt
```

In [ ]:
```python
num_points = 1000
delta = 1e-5
```

Random Line generated as target function

In [ ]:
```python
ax = np.random.uniform(-1, 1 + 1e-5)
bx = np.random.uniform(-1, 1 + 1e-5)
ay = np.random.uniform(-1, 1 + 1e-5)
by = np.random.uniform(-1, 1 + 1e-5)

f = np.array([bx - ax, by - ay])
```

In [ ]:
```python
def target_function(x, y):
    cross_product = (bx - ax)*(y - ay) - (by - ay)*(x - ax)

    if cross_product > 0:
        return -1
    else:
        return 1

ax
```

Out[ ]:
```
0.5098814203749296
```

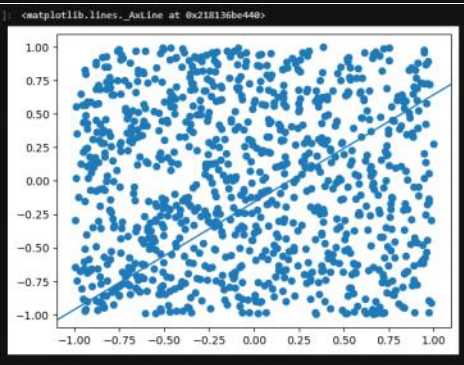
In [ ]:
```python
def sign_inner_product(w, x):
    return np.sign(np.dot(w, x))
```

In [ ]:
```python
x1 = np.random.uniform(-1, 1 + delta, num_points)
x2 = np.random.uniform(-1, 1 + delta, num_points)

xn = np.column_stack((x1, x2))
yn = [target_function(x, y) for (x, y) in xn]

plt.plot([i[0] for i in xn], [i[1] for i in xn], 'o')
plt.axline((ax, bx), (ay, by))
```

Out[ ]:
```
<matplotlib.lines._AxLine at 0x218136be440>
```



In [ ]:
```python
weights = np.zeros(2)
```

In [ ]:
```python
misclassified_points = list(zip(xn, yn))
random_index = np.random.randint(0, num_points)
misclassified_point = (xn[random_index], yn[random_index])
misclassified_point
```

Out[ ]:
```
(array([0.71854599, 0.99024622]), 1)
```

In [ ]:
```python
iterations = [weights]
```

In [ ]:
```python
def PLA(xn, yn):
    random_index = np.random.randint(0, num_points)
    misclassified_point = (xn[random_index], yn[random_index])

    while sign_inner_product(iterations[-1], misclassified_point[0]) == np.sign(misclassified_point[1]):
        random_index = np.random.randint(0, num_points)
        misclassified_point = (xn[random_index], yn[random_index])

    w = iterations[-1] + misclassified_point[1] * misclassified_point[0]
    iterations.append(w)
```

In [ ]:
```python
for i in range(100000):
    PLA(xn, yn)
```

In [ ]:
```python
iterations[-1]
```

Out[ ]:
```
array([-11.24364089,  11.81442367])
```

In [ ]:
```python
f
```

Out[ ]:
```
array([-0.26004582, -0.27305425])
```

In [ ]: