CS 156 a — Problem Set #1

Ishaan Mantripragada.

Problem 1. ***[d] (i) Not learning, (ii) Supervised Learning, (iii) Reinforcement Learning.*** *The first scenario does not show any learning because there is no algorithm or model that learns from known information. The classification system only relies on the coin specifications from the U.S. Mint. The second scenario describes a supervised learning scenario since an algorithm is trained on a labelled set of coins to determine the classification for other coins. The third scenario describes reinforcement learning since the computer does not start with any labelled data, but it learns from its moves and "penalizes" losing moves which is equivalent to adjusting its weights. Thus, the computer learns from moves that it makes itself.*

Problem 2. ***[a] (ii) and (iv).*** *Problem 2 is suited for Machine Learning because an algorithm can be developed by learning from past cases of credit card fraud to detect future cases. Supervised Learning could be used in this case. Problem 4 is also suited for Machine Learning because a computer can simulate traffic light times and adjust them based on their efficiency. Reinforcement Learning would be used in this case.*

Problem 3. ***[d]*** **$2/3$**. *There are four total possibilities of picking balls. The probability of picking a black ball first from either bag is* $(1/2)(1) + (1/2)(1/2) = 3/4$. *Therefore, there are 3 total possibilities for drawing a black ball first. The probability of then picking another black ball is 2/3, because there are two ways of picking from the bag with two blacks (which guarantees another black) and only one way of picking from the bag with one black and one white (which would result in a white ball).*

Problem 4. ***[b]*** $\mathbf{3.405 \times 10^{-4}}$. *The probability that* $\nu = 0$ *if we draw one sample is* $(1-\mu)^{10} = (0.45)^{10} \approx 3.405 \times 10^{-4}$.

Problem 5. ***[c]*** **0.289**. *The probability that one sample has* $\nu = 0$ *is* $1 - 3.405 \times 10^{-4}$. *The probability that this happens over 1000 trials is* $(1 - 3.405 \times 10^{-4})^{1000}$. *As a result, the probability that it happens at least once is* $1 - (1 - 3.405 \times 10^{-4})^{1000} \approx 0.289$.

Problem 6. ***[e].***

0	0	0	0	1	1	1	1
0	0	1	1	0	0	1	1
0	1	0	1	0	1	0	1

Table 1: All Possible y_n

[a] Score: $1 * 3 + 3 * 2 + 3 * 1 = 12$
[b] Score: $1 * 3 + 3 * 2 + 3 * 1 = 12$
[c] XOR(x_6*,*x_7*,*x_8*)* $= \{0, 0, 1\}$, *Score:* $1 * 3 + 3 * 2 + 3 * 1 = 12$
[d] !XOR(x_6*,*x_7*,*x_8*)* $= \{1, 1, 0\}$, *Score:* $1 * 3 + 3 * 2 + 3 * 1 = 12$
Therefore, [e] all are equivalent.

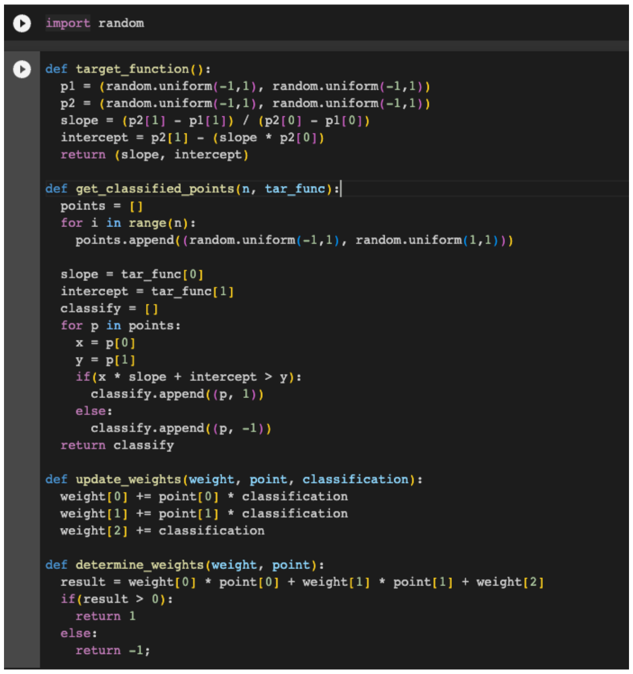
For Problems 7-10, please see images at the end of the document.

Problem 7. *[b].*

Problem 8. *[c].*

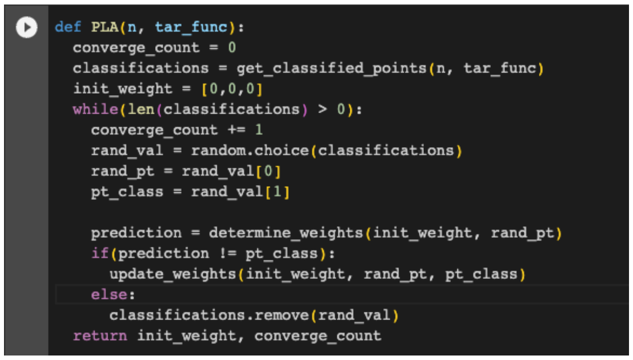
Problem 9. *[b].*

Problem 10. *[b].*

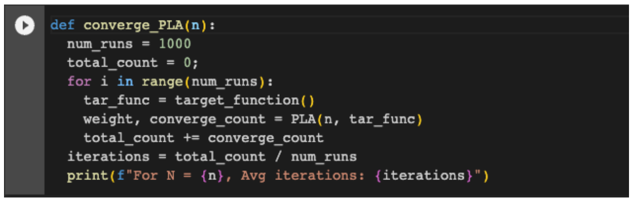


```python
import random
```

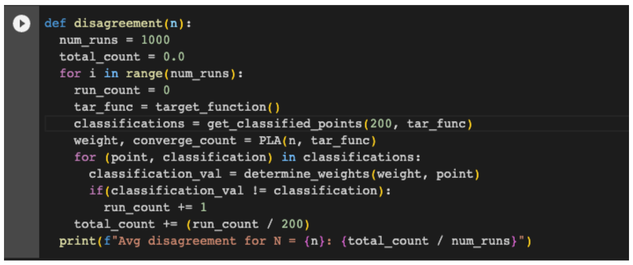
```python
def target_function():
  p1 = (random.uniform(-1,1), random.uniform(-1,1))
  p2 = (random.uniform(-1,1), random.uniform(-1,1))
  slope = (p2[1] - p1[1]) / (p2[0] - p1[0])
  intercept = p2[1] - (slope * p2[0])
  return (slope, intercept)

def get_classified_points(n, tar_func):
  points = []
  for i in range(n):
    points.append((random.uniform(-1,1), random.uniform(1,1)))

  slope = tar_func[0]
  intercept = tar_func[1]
  classify = []
  for p in points:
    x = p[0]
    y = p[1]
    if(x * slope + intercept > y):
      classify.append((p, 1))
    else:
      classify.append((p, -1))
  return classify

def update_weights(weight, point, classification):
  weight[0] += point[0] * classification
  weight[1] += point[1] * classification
  weight[2] += classification

def determine_weights(weight, point):
  result = weight[0] * point[0] + weight[1] * point[1] + weight[2]
  if(result > 0):
    return 1
  else:
    return -1;
```



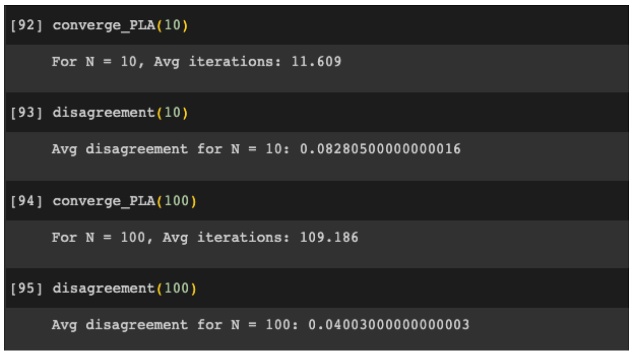
```python
def PLA(n, tar_func):
  converge_count = 0
  classifications = get_classified_points(n, tar_func)
  init_weight = [0,0,0]
  while(len(classifications) > 0):
    converge_count += 1
    rand_val = random.choice(classifications)
    rand_pt = rand_val[0]
    pt_class = rand_val[1]

    prediction = determine_weights(init_weight, rand_pt)
    if(prediction != pt_class):
      update_weights(init_weight, rand_pt, pt_class)
    else:
      classifications.remove(rand_val)
  return init_weight, converge_count
```



```python
def converge_PLA(n):
  num_runs = 1000
  total_count = 0;
  for i in range(num_runs):
    tar_func = target_function()
    weight, converge_count = PLA(n, tar_func)
    total_count += converge_count
  iterations = total_count / num_runs
  print(f"For N = {n}, Avg iterations: {iterations}")
```



```python
def disagreement(n):
  num_runs = 1000
  total_count = 0.0
  for i in range(num_runs):
    run_count = 0
    tar_func = target_function()
    classifications = get_classified_points(200, tar_func)
    weight, converge_count = PLA(n, tar_func)
    for (point, classification) in classifications:
      classification_val = determine_weights(weight, point)
      if(classification_val != classification):
        run_count += 1
    total_count += (run_count / 200)
  print(f"Avg disagreement for N = {n}: {total_count / num_runs}")
```



```
[92] converge_PLA(10)

     For N = 10, Avg iterations: 11.609

[93] disagreement(10)

     Avg disagreement for N = 10: 0.08280500000000016

[94] converge_PLA(100)

     For N = 100, Avg iterations: 109.186

[95] disagreement(100)

     Avg disagreement for N = 100: 0.04003000000000003
```