CS 156a - Problem Set 1

Samuel Patrone, 2140749

October 10, 2022

The following notebook is publicly available at the following link.

Contents

Problem 1

Answer: [d]

Derivation:

(i): the target function is known in advance (from the specifications given from US Mint). Nothing is learned by the machine.

(ii): a training set of data with known output is given in advance to the algorithm in order to learn the target function itself. This is an example of supervised learning.

(iii): a game is the classical example of reinforcement learning, in which an evaluation of the output (grading) is provided to the machine in order to learn and adjust to develop the best strategy.

Problem 2

Answer: [a]

Derivation:

A generic problem, in order to be best suited for ML, needs three requisites:

- a pattern exists
- it cannot be pinned down mathematically
- we have DATA on it

In the list of problem presented

(i): the problem of primes numbers identification can be pinned down mathematically.

(ii): detecting potential fraud in credit cards has all the three characteristics described above -> ML suited

(iii): the law of mechanics can describe the equation of motion of any object (under certain conditions). Again, the problem can be solved analytically.

(iv): optimal cycle for traffic lights has again all the three characteristics described above -> ML suited

Problem 3

Answer: [d]

Derivation:

To solve this problem, we can use the standard formula for conditional probability, which states that

$$P(A|B) = \frac{P(A \cap B)}{P(B)}\,.$$

where

- A is the event of picking up a black ball in our second draw
- B is the event of picking up a black ball in our first draw
- $P(A|B)$ is the probability of that the second ball is black given that the first ball is black, i.e. what the problem requires us to compute
- $P(A \cap B) = \frac{1}{2}$ is the probability of that we pick two black balls, i.e. the probability of choosing the bag with two black balls inside
- $P(B) = \frac{3}{4}$ is the probability of picking up a black ball in our first draw

Applying the equation above, we have that

$$P(A|B) = \frac{1}{2}\frac{4}{3} = \frac{2}{3}.$$

Problem 4

Answer: [b]

Derivation:

The probability to get no red marbles in one draw is $\bar{\mu} = (1 - \mu) = 0.45$. In one sample, we draw 10 marbles independently, the probability to get all green marbles, i.e. $\nu = 0$, is just:

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

Problem 5

Answer: [c]

Derivation:

The probability of getting no red marbles ($\nu = 0$) in one sample was computed in the previous equation. The complementary probability of getting $\nu \neq 0$ is therefore $P(\nu \neq 0) = 1 - \bar{\mu}^{10}$. For one thousand independent samples, the probability of getting $\nu \neq 0$ in all the samples is therefore

$$P(\nu \neq 0)^{1000} = 0.711.$$

Finally, the probability of getting at least one sample with all green marbles is

$$1 - P(\nu \neq 0)^{1000} = 0.289.$$

Problem 6

Answer: [e]

Derivation:

The 8 possible functions h_i on $\mathcal{X} = \{\mathbf{x}_6, \mathbf{x}_7, \mathbf{x}_8\}$, where

$\mathbf{x}_6 = \{1, 0, 1\}$
$\mathbf{x}_7 = \{1, 1, 0\}$
$\mathbf{x}_8 = \{1, 1, 1\}$,

are the following:

$$\begin{aligned} h_1[\mathcal{X}] &= \{1, 1, 1\} \\ h_2[\mathcal{X}] &= \{0, 1, 1\} \\ h_3[\mathcal{X}] &= \{1, 0, 1\} \\ h_4[\mathcal{X}] &= \{1, 1, 0\} \\ h_5[\mathcal{X}] &= \{0, 0, 1\} \\ h_6[\mathcal{X}] &= \{0, 1, 0\} \\ h_7[\mathcal{X}] &= \{1, 0, 0\} \\ h_8[\mathcal{X}] &= \{0, 0, 0\}. \end{aligned} \tag{1}$$

Let's explore the hypothesis function g scores for different cases, where the score $\Sigma(g)$ is defined as the following:

$\Sigma(g)$ = (# 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.

(a) $g_1[\mathcal{X}] = \{0, 0, 1\}$

It agrees on: - all of 3 points for h_1 - exactly 2 points for h_2, h_3, h_4 - exactly 1 point for h_5, h_6, h_7

Therefore, $\Sigma(g_1) = 3 \times 1 + 2 \times 3 + 1 \times 3 = 12$.

(b) $g_2[\mathcal{X}] = \{1, 1, 0\}$

It agrees on: - all of 3 points for h_8 - exactly 2 points for h_2, h_3, h_8 - exactly 1 point for h_4, h_3, h_2

Therefore, $\Sigma(g_2) = 3 \times 1 + 2 \times 3 + 1 \times 3 = 12$.

(c) $g_3[\mathcal{X}] = \{0, 0, 1\}$.

It agrees on: - all of 3 points for h_5 - exactly 2 points for h_2, h_3, h_8 - exactly 1 point for h_1, h_6, h_7

Therefore, $\Sigma(g_3) = 12$.

(d) $g_4[\mathcal{X}] = \{1, 1, 0\}$.

It agrees on: - all of 3 points for h_4 - exactly 2 points for h_1, h_6, h_7 - exactly 1 point for h_2, h_3, h_8

Therefore, $\Sigma(g_4) = 12$.

Hence, all the hypothesis are equivalent, given the above score formula.

Problem 7 - 10: The Perceptron Learning Algorithm

Answers: [b] , [c] , [b] , [b]

Code:

[17]:
```
import numpy as np
import matplotlib.pyplot as plt

N1=10
N2=100
Nruns=1000

def gen_points_2d(N,xleft=-1,xright=1,yleft=-1,yright=1):
    pts=[[np.random.uniform(xleft,xright),np.random.uniform(yleft,yright)] for i␣
 ↪in range(N)]
    return pts

def extract_x(lst):
    return list(list(zip(*lst))[0])
def extract_y(lst):
    return list(list(zip(*lst))[1])

def gen_line():
    x1,x2,y1,y2=np.random.uniform(-1,1),np.random.uniform(-1,1),np.random.
 ↪uniform(-1,1),np.random.uniform(-1,1)
    m=(y2-y1)/(x2-x1)
    b=(y1*x2-y2*x1)/(x2-x1)
    return m,b

def f(x,m,b):
    return m*x+b

def gen_input(N_pts,m,b,plot=True):
    data=gen_points_2d(N_pts)
    output=[]
    x0=extract_x(data)
    y0=extract_y(data)
    for i in range(N_pts):
        if(f(x0[i],m,b)>y0[i]): output.append(1.)
        else: output.append(-1.)
    if(plot==True):
        col=[]
        for i in range(len(output)):
            if(output[i]>0): col.append('red')
            else: col.append('green')
        x=np.linspace(-1,1,100)
        plt.plot(x,f(x,m,b),color='blue')
```

```
        plt.scatter(x0,y0,color=col)
        plt.xlim([-1, 1])
        plt.ylim([-1, 1])
    return np.array(data), np.array(output)

def h(w,data,bias):
    return np.sign(np.dot(w,data.T)+bias)

def validating(m,b,w,bias,Nval=1000):
    data,output=gen_input(Nval,m,b,plot=False)
    g=h(w,data,bias)
    testg=(g==output)
    misclassified=len(np.where(testg==False)[0])
    return misclassified/Nval

def PLA_run(N_pts, plot=False):
    #generating linearly separable points
    m,b=gen_line()
    data,output=gen_input(N_pts,m,b,plot)

    #initialization before learning
    w=np.zeros(2)
    bias=0
    converged=False
    iterations=0
    #learning algorithm
    while(converged==False):
        g=h(w,data,bias)
        testg=(g==output)
        misclassified=np.where(testg==False)[0]
        if(len(misclassified)>0):
            i=misclassified[0]
            w+=output[i]*data[i]
            bias+=output[i]
            iterations+=1
        g=h(w,data,bias)
        converged=np.all(g==output)
    #validation
    prob=validating(m,b,w,bias)
    if(plot==True):
        x=np.linspace(-1,1,100)
        plt.plot(x,f(x,-w[0]/w[1],-bias/w[1]),color='blue',linestyle='dashed')
        plt.show
    return iterations,prob

def average(N_pts,N_runs):
    average_iter=0
```

```
    average_prob=0
    for i in range(N_runs):
        average_iter+=PLA_run(N_pts)[0]
        average_prob+=PLA_run(N_pts)[1]
    print("#######")
    print("Here's the result for N_runs=",N_runs,"with N_pts=",N_pts)
    print("Average iterations for convergence:",average_iter/N_runs)
    print("Average misclassification rate:",average_prob/N_runs)
```

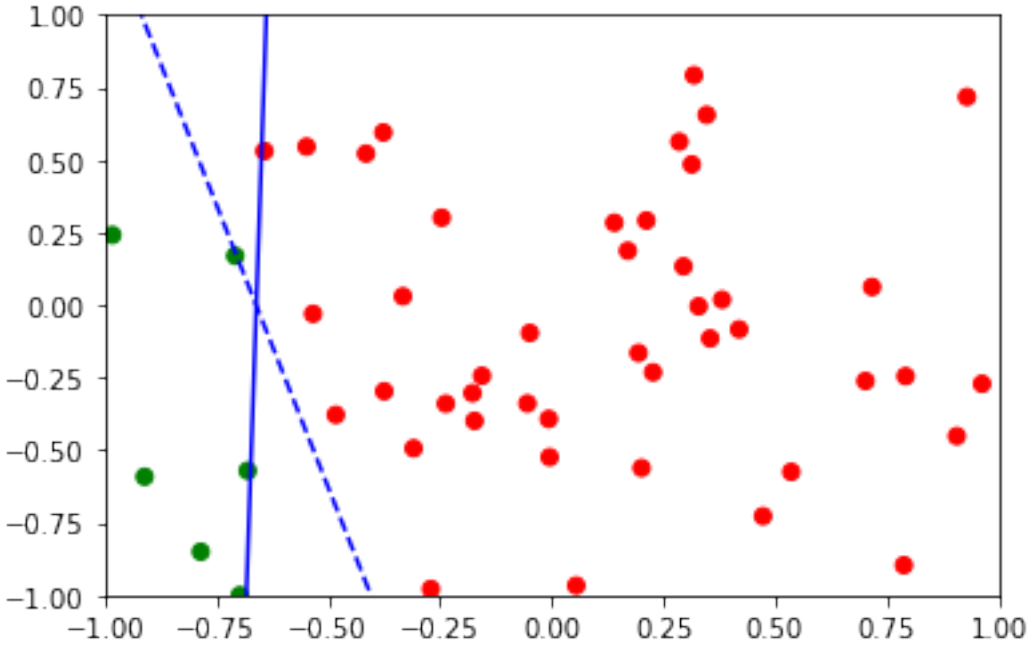
[217]:
```
average(10,1000)
average(100,1000)
```

```
#######
Here's the result for N_runs= 1000 with N_pts= 10
Average iterations for convergence: 11.311
Average misclassification rate: 0.10873300000000007
#######
Here's the result for N_runs= 1000 with N_pts= 100
Average iterations for convergence: 198.46
Average misclassification rate: 0.013101999999999961
```

[39]:
```
#Here's a plotted example with 50 points. The solid line is the target function␣
↪f while the dashed line is the g obtained by the Perceptron Learning Algorithm.
↪ In the output tuple, the first number is the number of iterations needed to␣
↪converge, the second number represents a numerical estimate of the␣
↪misclassification rate.

PLA_run(50,plot=True)
```

[39]:
```
(35, 0.067)
```


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00