Assignment -2 Submission

Name: GOVIND AJITH KUMAR
VID: 11669948 8

2.1

$$\epsilon(M, N, \delta) = \sqrt{\frac{1}{2N} \ln \frac{2M}{\delta}}, \quad \delta = 0.03 \text{ (given)}$$

(a) for $M = 1$,
for $\epsilon \leq 0.05$

$$\Rightarrow \sqrt{\frac{1}{2N} \times \ln\left(\frac{2\times 1}{0.03}\right)} \leq 0.05$$

$$\Rightarrow \frac{1}{2N} \ln\left(\frac{200}{3}\right) \leq 0.0025 \text{ (squaring both the sides)}$$

$$\Rightarrow \frac{1}{2N} \leq \frac{0.0025}{4.199} \Rightarrow \frac{1}{2N} \leq 5.953 \times 10^{-4}$$

$$\Rightarrow 2N \geq 1679.825 \Rightarrow N \geq 839.91$$

$$\approx \boxed{N \geq 840}$$

(b) for $M = 100$,
for $\epsilon \leq 0.05$

$$\Rightarrow \frac{1}{2N} \ln\left(\frac{200 \times 100}{3}\right) \leq 0.0025$$

$$\Rightarrow \frac{1}{2N} \leq \frac{0.0025}{8.804} \Rightarrow 2N \geq 3521.6$$

$$\Rightarrow N \geq 1760.8$$

$$\approx \boxed{N \geq 1761}$$

(c) for $M = 10{,}000$, for $\epsilon \leq 0.05$

$$\Rightarrow \frac{1}{2N} \ln\left(\frac{200}{2} \times 19{,}000\right) \leq 0.0025$$

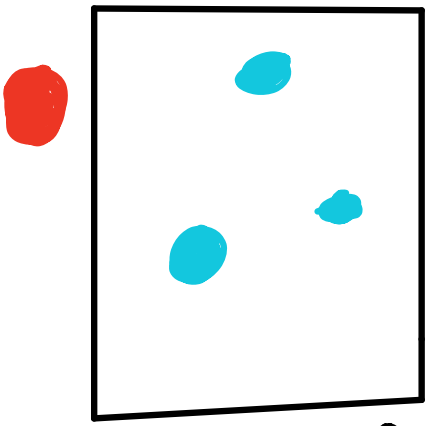
$$\Rightarrow \frac{1}{2N} \leq \frac{0.0025}{13.41} \Rightarrow 2N \geq 5{,}364.018$$

$$\Rightarrow N \geq 2682.009$$

$$\approx \boxed{N \geq 2682}$$

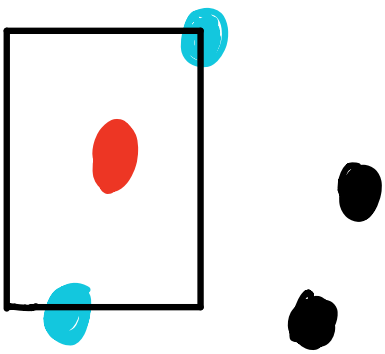
Reference : LFD Text Book

2.2 For a learning model of positive rectangles, the growth function $m_H(4) = 2^4$ can be achieved because a rectangle can be drawn for all possible combinations



But for 5 points, some point will always be inside the rectangle

Hence $m_H(5) < 2^5$. This limits the ways in which we can create the dichotomies



$\therefore m_H(4) = 2^4$

$m_H(5) < 2^5$

Hence for positive rectangle the

$d_{VC} = 4.$

Reference : LFD Text Book

2.11 $m_H(N) = N+1$, $d_{VC} = 1$

Number of training samples = 100

for confidence of 90%

We know $E_{out}(g) \leq E_{in}(g) + \sqrt{\frac{8}{100} \ln \frac{4 m_H(2N)}{\delta}}$

for $N = 100$:

$$E_{out}(g) \leq E_{in}(g) + \sqrt{\frac{8}{100} \ln \frac{4(2 \cdot 100 + 1)}{0.1}}$$

$$= \boxed{E_{in}(g) + 0.848}$$

Now, when $N = 10,000$

$$E_{out}(g) \leq E_{in}(g) + \sqrt{\frac{8}{100} \cdot \ln \frac{4(2 \cdot 10000 + 1)}{0.1}}$$

$$= \boxed{E_{in}(g) + 0.104}$$

Reference : LFD Text Book

2.12 CODE ATTACHED

Reference : LFD Text Book

2.23 CODE ATTACHED

Reference : LFD Text Book and lecture

2.4 To prove:

$$B(N,k) \geq \sum_{i=0}^{k-1} \binom{N}{i}.$$

Problem states to limit the number of -1's.

As per Sauer's Lemma:

$$B(N,k) \leq \sum_{i=0}^{k-1} \binom{N}{i}$$

Dichotomies that contain 0 -1s can be written as: $\binom{N}{0}$

Similarly dichotomies that contain (i) -1s can be $\binom{N}{i}$.

Thus adding all the possibilities of dichotomies we get:

$$\binom{N}{0} + \binom{N}{1} + \cdots \binom{N}{1}$$

$$\Rightarrow \sum_{i=0}^{k-1} \binom{N}{i}$$ is the total number of dichotomies that we have.

But this does not shatter any subset of k variables. Hence

$$B(N,k) \geq \sum_{i=0}^{k-1} \binom{N}{i}$$. But from Sauer's Lemma:

$$B(N,k) \leq \sum_{i=0}^{k-1} \binom{N}{i}$$. Hence we can conclude that:

$$\boxed{B(N,k) = \sum_{i=0}^{k-1} \binom{N}{i}}$$

Reference : LFD Text Book

8 **to prove:** VC dimension of a perceptron in R^d is $d+1$.

There are a couple of steps to be proven to prove this:

→ the perceptron can shatter $d+1$ points

→ the hypothesis H cannot shatter points $\geq d+2$ i.e; $d_{VC} \leq d+1$

Let the input space be $x = [x_1, x_2 \cdots x_d, 1]^T$

As per perceptron learning algorithm:

$$\text{sign}(x_n^T w) = y_n \; ; \; y_n = \{+1, -1\} \quad \text{(1)}$$

Hence,

$$\begin{cases} \text{sign}(x_1^T w) = y_1 \\ \vdots \\ \text{sign}(x_{d+1}^T w) = y_{d+1} \end{cases}$$

This can be written as:

$$\begin{bmatrix} -x_1^T- \\ -x_2^T- \\ \vdots \\ -x_{d+1}^T- \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 0 & \cdots & 0 \\ & & \ddots & & \\ 1 & 0 & \cdots & & 1 \end{bmatrix}}_{M} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{d+1} \end{bmatrix}$$

$$= \begin{bmatrix} \pm 1 \\ \pm 1 \\ \vdots \\ \pm 1 \end{bmatrix} \rightarrow \text{from (1) we know this}$$

Here, the matrix 'm' is 'd+1' in dimension and is a square matrix. Therefore for any given y, we can get w and hence we can say that $d_{VC} \geq d+1$.

Now, remember that our perceptron is a d-dimensional learning algorithm. We need to prove that we cannot shatter any set of d+2 points for a d-dimensional learning algorithm.

Say we have d+2 points, $x_1, x_2 \cdots x_{d+1}, x_{d+2}$

We can write that:

$$X_{d+2} = \sum_{j=1}^{d+1} \alpha_j x_j$$

A dichotomy can be constructed as follows:

$$y_i = \begin{cases} sign(\alpha_i) & i = 1, 2 \ldots d+1 \\ -1 & i = d+2 \end{cases} \quad \text{---(2)}$$

Now, we know that $\text{sign}(d_i) = \text{sign}(w^T x_i)$

So, if $i = d+2$:

$$y_{d+2} = \text{sign}(w^T x_{d+2}),$$ which is equal to -1 (from ②). So, the VC dimension of perceptron of dimension d is at most $d+1$.

$\therefore d_{VC} \leq d+1$ & $d_{VC} > d+1$

$\Rightarrow \boxed{d_{VC} = d+1}$ Hence proved

Reference : http://www.stat.ucdavis.edu/~chohsieh/teaching/ECS171_Winter2018/lecture8.pdf

---

7 To show that k is a break point for H:

→ correct answer: Show H cannot shatter any set of k points $x_1, x_2 \ldots x_k$

point ① → This is not correct because if a set of k points $(x_1, x_2 \ldots x_k)$ can be shattered, then k is NOT a break point for H

point ② This is an extreme condition, but we cannot guarantee that any random set of K points can be shattered by H. In any case even if that happens, K is NOT a breakpoint for H.

point ③ By definition again, this cannot be right.

point ④ By definition, if H cannot shatter any set of K points $x_1, x_2 \cdots x_K$, then K is a breakpoint for H.

BiasVarianceDecomposition

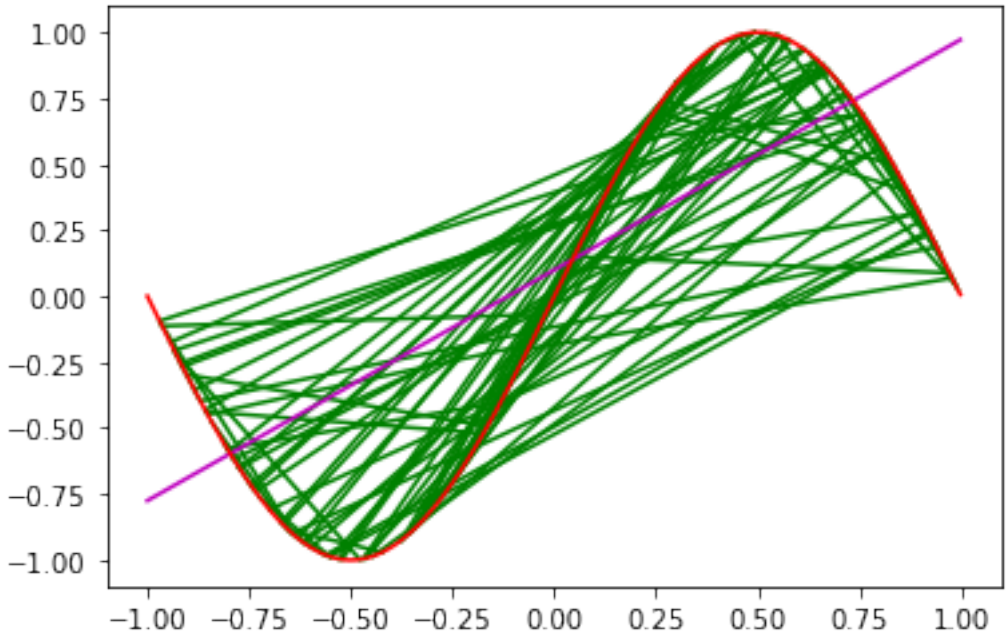
October 12, 2020

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from random import seed
from random import randint
from statistics import mean
seed(25)
```

[2]:
```
N = 500
x = np.arange(-1,1,(1/N))   # start,stop,step
y = np.sin(np.pi*x)
```

[3]:
```
def findYCoord(point1,point2,x_point):
    #find slope
    slope = (point2[1]-point1[1])/(point2[0]-point1[0])
    y_intercept = point1[1] + slope*(x_point-point1[0])
    return y_intercept
plt.figure()
#a diff list which contains the difference between the predicited and the sin␣
 ↪intercept
all_points_dic={}
bias_list = []
variance_list = []
for i in x:
    all_points_dic[i]=[]
#for loop to draw out random lines after selecting two random points from the␣
 ↪sin graph
for every_point in range(100):
    try:
        #finding a random number
        point_1_idx = randint(0,2*(N)-1)
        point_2_idx = randint(0,2*(N)-1)
        #indexing out the x coordinate at the random numbers index
        x_coord_1 = x[point_1_idx]
        x_coord_2 = x[point_2_idx]
        #indexing out the y coordinate at the random numbers index as per the␣
 ↪sin value
        y_coord_1 = np.sin(np.pi*x_coord_1)
```

```
        y_coord_2 = np.sin(np.pi*x_coord_2)
        #preparing the x and the y coordinates for plotting
        x_line = [x_coord_1,x_coord_2]
        y_line = [y_coord_1,y_coord_2]
        for i in x:
            point1 = (x_line[0],y_line[0])
            point2 = (x_line[1],y_line[1])
            y_coord_found = findYCoord(point1,point2,i)
            all_points_dic[i].append(y_coord_found)
        #plotting the graph
        plt.plot(x_line,y_line,'g')
    except:
        pass
#plotting out the sin curve, in red
y_means = []
for k,v in all_points_dic.items():
    y_means.append(mean(v))
    #finding bias
    f = np.sin(np.pi*k)
    mean_v = mean(v)
    bias_list.append((mean_v-f)**2)
    #finding variance
    for e in v:
        variance_list.append(e**2-mean_v**2)
plt.plot(x,y_means,'m-')
plt.plot(x,y,'r')
plt.show()
#printing bias
final_bias = np.mean(bias_list)
print('Bias is:', final_bias)
#printing variance
final_variance = np.mean(variance_list)
print('Variance is:', final_variance)
#printing out-of-sample error
out_of_sample_error = final_bias + final_variance
print('Out-of-sample error is:', out_of_sample_error)
```



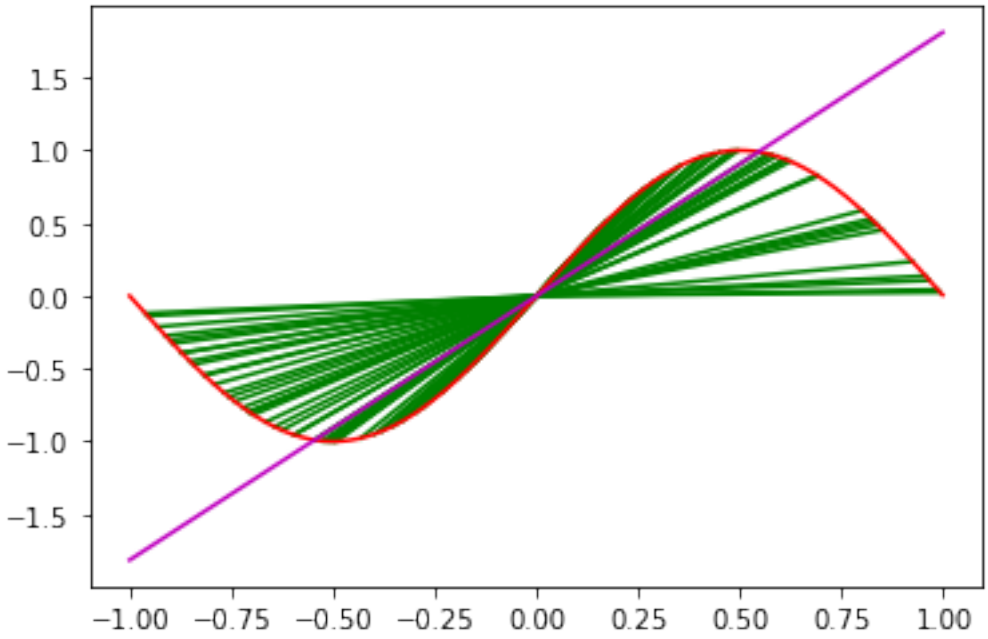

```
Bias is: 0.2079040047664947
Variance is: 1.8874220950524334
Out-of-sample error is: 2.0953260998189283
```

[7]:
```
a_bar_list = []
b_bar_coord = []
def findYCoord(point1,point2,x_point):
    global b_bar_coord
    global a_bar_list
    #find slope
    slope = (point2[1]-point1[1])/(point2[0]-point1[0])
    a_bar_list.append(slope)
    y = point1[1] + slope*(x_point-point1[0])
    b_bar_coord.append(y)
    return y
plt.figure()
#a diff list which contains the difference between the predicited and the sin␣
 ↪intercept
all_points_dic={}
bias_list = []
variance_list = []
for i in x:
    all_points_dic[i]=[]
#for loop to draw out random lines after selecting two random points from the␣
 ↪sin graph
for every_point in range(100):
```

```
    try:
        #finding a random number
        point_1_idx = randint(0,2*(N)-1)
        point_2_idx = randint(0,2*(N)-1)
        #indexing out the x coordinate at the random numbers index
        x_coord_1 = 0
        x_coord_2 = x[point_2_idx]
        #indexing out the y coordinate at the random numbers index as per the␣
 ↪sin value
        y_coord_1 = 0
        y_coord_2 = np.sin(np.pi*x_coord_2)
        #preparing the x and the y coordinates for plotting
        x_line = [x_coord_1,x_coord_2]
        y_line = [y_coord_1,y_coord_2]
        for i in x:
            point1 = (x_line[0],y_line[0])
            point2 = (x_line[1],y_line[1])
            y_coord_found = findYCoord(point1,point2,i)
            all_points_dic[i].append(y_coord_found)
        #plotting the graph
        plt.plot(x_line,y_line,'g')
    except:
        pass
#plotting out the sin curve, in red
plt.plot(x,y,'r')
y_means = []
for k,v in all_points_dic.items():
    y_means.append(mean(v))
plt.plot(x,y_means,'m-')
a_bar = mean(a_bar_list)
bias_list = []
variance_list = []
for k,v in all_points_dic.items():
    bias_list.append(((k*a_bar)-(np.sin(np.pi*k)))**2)
final_bias = np.mean(bias_list)
print('Bias is:', final_bias)
for k,v in all_points_dic.items():
    for e in v:
        variance_list.append(((e-a_bar)**2)*(k**2))
final_variance = np.mean(variance_list)
print('Variance is:', final_variance)
#printing out-of-sample error
out_of_sample_error = final_bias + final_variance
print('Out-of-sample error is:', out_of_sample_error)
```

```
Bias is: 0.44264448697577735
Variance is: 1.9600153275839263
```

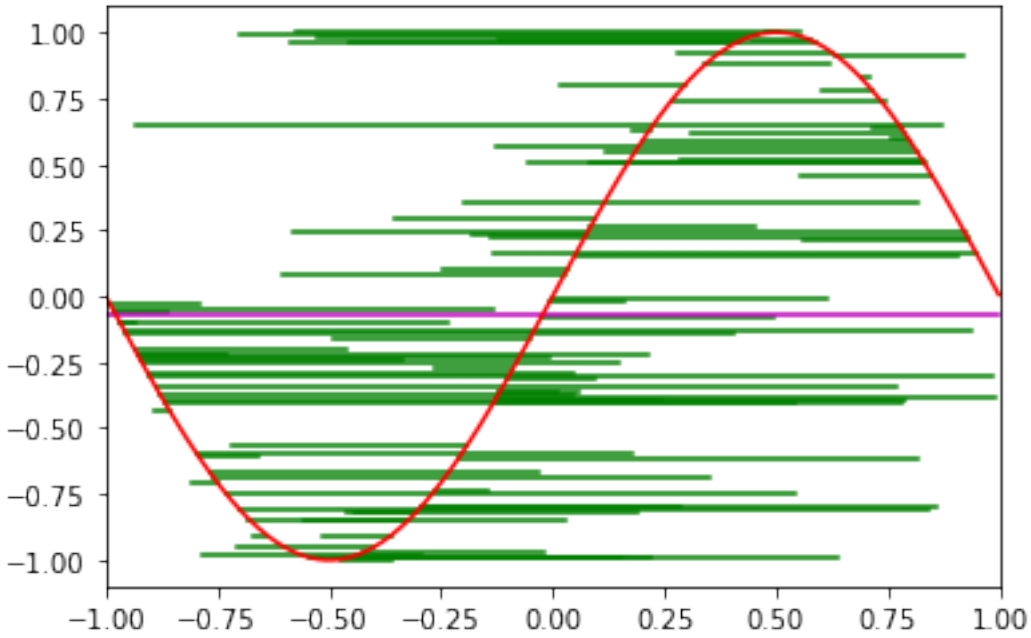
```
Out-of-sample error is: 2.4026598145597036
```



[5]:
```
def findYCoord(point1,point2,x_point):
    #find slope
    slope = (point2[1]-point1[1])/(point2[0]-point1[0])
    y_intercept = point1[1] + slope*(x_point-point1[0])
    return y_intercept
plt.figure()
#a diff list which contains the difference between the predicited and the sin␣
 ↪intercept
all_points_dic={}
bias_list = []
variance_list = []
for i in x:
    all_points_dic[i]=[]
#for loop to draw out random lines after selecting two random points from the␣
 ↪sin graph
for every_point in range(100):
    try:
        #finding a random number
        point_1_idx = randint(0,2*(N)-1)
        point_2_idx = randint(0,2*(N)-1)
        #indexing out the x coordinate at the random numbers index
        x_coord_1 = x[point_1_idx]
        x_coord_2 = x[point_2_idx]
```

```
        #indexing out the y coordinate at the random numbers index as per the␣
 ↪sin value
        y_coord_1 = np.sin(np.pi*x_coord_1)
        y_coord_2 = np.sin(np.pi*x_coord_1)
        #preparing the x and the y coordinates for plotting
        x_line = [x_coord_1,x_coord_2]
        y_line = [y_coord_1,y_coord_2]
        for i in x:
            point1 = (x_line[0],y_line[0])
            point2 = (x_line[1],y_line[1])
            y_coord_found = findYCoord(point1,point2,i)
            all_points_dic[i].append(y_coord_found)
        #plotting the graph
        plt.xlim(-1,1)
        plt.plot(x_line,y_line,'g')
    except:
        pass
#plotting out the sin curve, in red
y_means = []
for k,v in all_points_dic.items():
    y_means.append(mean(v))
    #finding bias
    f = np.sin(np.pi*k)
    mean_v = mean(v)
    bias_list.append((mean_v-f)**2)
    #finding variance
    for e in v:
        variance_list.append((e-mean_v)**2 )

plt.plot(x,y_means,'m-')
plt.plot(x,y,'r')
plt.show()
#printing bias
final_bias = np.mean(bias_list)
print('Bias is:', final_bias)
#printing variance
final_variance = np.mean(np.array(variance_list)**2)
print('Variance is:', final_variance)
#printing out-of-sample error
out_of_sample_error = final_bias + final_variance
print('Out-of-sample error is:', out_of_sample_error)
```




```
Bias is: 0.5053130009894711
Variance is: 0.28331995919518826
Out-of-sample error is: 0.7886329601846593
```

[ ]:

LFD Problem 2.12

October 12, 2020

[26]:
```
import numpy as np
d_vc = 10
e = 0.05
delta = 0.05
flag = False
x = 1000
while(True):
    N  = (8/(0.05**2))*(np.log((4*(((2*x)**d_vc)+1))/0.05))
    print(N)
    if (N<=x):
          break
    else:
        x = N
print('Hence, the N value we want is :', N)
```

```
257251.363936303
434853.0815903086
451651.62731454166
452864.5206285506
452950.34023365
452956.40378480166
452956.83215921914
452956.8624225618
452956.8645605733
452956.8647116172
452956.86472228804
452956.8647230419
452956.8647230951
452956.8647230988
452956.8647230991
452956.8647230992
452956.8647230992
Hence, the N value we want is : 452956.8647230992
```

[ ]: