P.S. 4

Monday, October 23, 2023 1:38 PM

1)

```
import numpy as np
import matplotlib.pyplot as plt
```

MagicPython

Problem 1

```
epsilon = 0.05
dvc = 10
current_N = 1
```

MagicPython

```
def get_omega(N, dvc, delta = 0.05):
    return ((8 / N) * np.log( (4 * (2 * N) ** dvc) / (delta) )) ** 0.5
```

MagicPython

```
while get_omega(current_N, dvc) >= epsilon:
    current_N += 1

current_N
```

MagicPython

```
452957
```

From the above code, we see the closest answer choice is d.

2)

Problem 2

```
dvc_fixed = 50
delta_fixed = 0.05
large_N = np.linspace(start=3, stop=10000, dtype="float64")
```

0.0s

```
log_mh = lambda N, dvc, constant: dvc * np.log(constant * N)
```

0.0s

```
vc_bound = lambda N: np.sqrt((8 / N) * (np.log(4) + log_mh(N, dvc_fixed, 2) - np.log(delta_fixed)))
```

0.0s

```
radermacher_bound = lambda N: np.sqrt((2 / N) * (np.log(2 * N) + log_mh(N, dvc_fixed, 1))) + np.sqrt((2 / N) * np.log(1 / delta_fixed)) + 1/N
```

0.0s

```
parrando_bound = lambda N: 1 / N + np.sqrt((np.log(6) + log_mh(N, dvc_fixed, 2) - np.log(delta_fixed) + 1)/ N)
```

0.0s

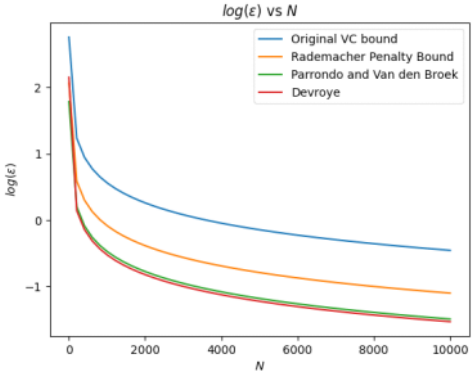
```
devroye_bound = lambda N: (1 / (N-2)) + np.sqrt(((np.log(4) + log_mh(N, dvc_fixed, N) - np.log(delta_fixed)) / (2 * N - 4)) + (1 / (N - 2)**2))
```

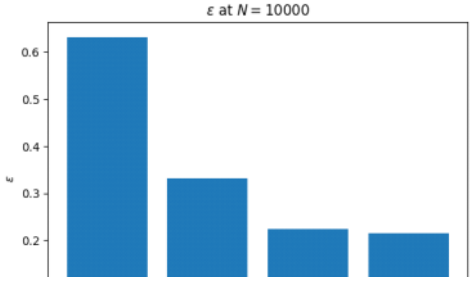
0.0s

We plot N against log(epsilon) to accentuate the differences between the graph.

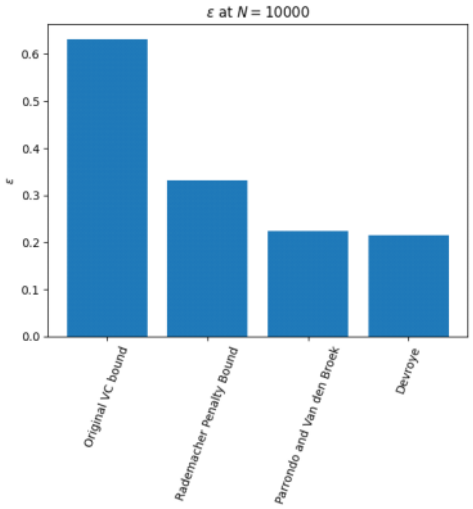
```
epsilons = [vc_bound(large_N),
            radermacher_bound(large_N),
            parrando_bound(large_N),
            devroye_bound(large_N)]

bounds = ["Original VC bound",
          "Rademacher Penalty Bound",
          "Parrondo and Van den Broek",
          "Devroye"]

plt.title("$log(\epsilon)$ vs $N$")
plt.xlabel("$N$")
plt.ylabel("$log(\epsilon)$")
for i in range(4):
    plt.plot(large_N, np.log(epsilons[i]))
    plt.legend(bounds)
plt.show()


N = 10000
epsilons = [vc_bound(N),
            radermacher_bound(N),
            parrando_bound(N),
            devroye_bound(N)]

bounds = ("Original VC bound",
          "Rademacher Penalty Bound",
          "Parrondo and Van den Broek",
          "Devroye")

plt.title("$\epsilon$ at $N=10000$")
plt.ylabel("$\epsilon$")
plt.xticks([0, 1, 2, 3], bounds, rotation=70)

plt.bar(np.arange(4), epsilons)
plt.show()
```

0.4s





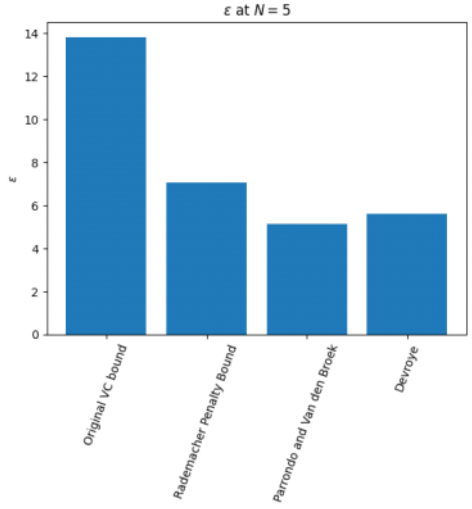
```
        parrando_bound(N),
        devroye_bound(N)]

bounds = ("Original VC bound",
          "Rademacher Penalty Bound",
          "Parrondo and Van den Broek",
          "Devroye")

plt.title("$\epsilon$ at $N=10000$")
plt.ylabel("$\epsilon$")
plt.xticks([0, 1, 2, 3], bounds, rotation=70)

plt.bar(np.arange(4), epsilons)
plt.show()
```
[11] ✓ 0.4s



After plotting each error bound and looking at the error at N=10000, we see the smallest error is produced by devroye, so the answer is d.

3)

```
N = 5
epsilons = [vc_bound(N),
            rademacher_bound(N),
            parrando_bound(N),
            devroye_bound(N)]

bounds = ("Original VC bound",
          "Rademacher Penalty Bound",
          "Parrondo and Van den Broek",
          "Devroye")

plt.title("$\epsilon$ at $N=5$")
plt.ylabel("$\epsilon$")
plt.xticks([0, 1, 2, 3], bounds, rotation=70)

plt.bar(np.arange(4), epsilons)
plt.show()
```
[12] ✓ 0.1s



looking at the error bound at N=5 we see the smallest error is produced by parrando, so the answer is c.

4)

```
import numpy as np
import matplotlib.pyplot as plt
```
[31] ✓ 0.0s

```
f = lambda x: np.sin(x)
```

4)

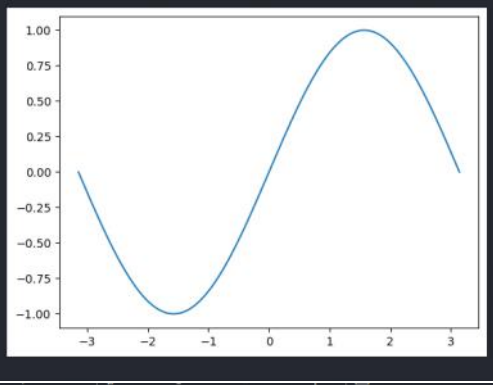
```
import numpy as np
import matplotlib.pyplot as plt
```
[31] ✓ 0.0s

```
f = lambda x: np.sin(x)
```
[32] ✓ 0.0s

```
x = np.pi * np.linspace(-1, 1, 5000)
y = f(x)
```
[33] ✓ 0.0s

```
plt.plot(x, y)
```
[34] ✓ 0.1s

```
[<matplotlib.lines.Line2D at 0x203fad79330>]
```




```
random_indices = np.random.choice(np.linspace(0, len(x) - 1, len(x)), 2)
training_set_x = np.array([x[int(random_indices[0])], x[int(random_indices[1])]])
training_set_y = np.array([y[int(random_indices[0])], y[int(random_indices[1])]])
```
[35] ✓ 0.0s

```
def calc_mean_square(m, train_x, pred_y):
    actual_y = []
    for x in train_x:
        actual_y.append(m * x)

    return np.square(np.subtract(actual_y, pred_y)).mean()
```
[36] ✓ 0.0s

```
mean_squares = []
```
[37] ✓ 0.0s

```
hypothesis_m = np.linspace(-5, 5, 1000)

for m in hypothesis_m:
    mean_squares.append((m, calc_mean_square(m, training_set_x, training_set_y)))
```
[38] ✓ 0.0s

```
def calc_min_mean_square(mean_square_arr):
    min_mean_square = float('inf')
    min_m = float('inf')

    for m, mean_square in mean_square_arr:
        if mean_square < min_mean_square:
            min_mean_square = mean_square
            min_m = m

    return min_m
```
[39] ✓ 0.0s

```
mean_squares_arr = []
```
[40] ✓ 0.0s

```
def run_trial():
    random_indices = np.random.choice(np.linspace(0, len(x) - 1, len(x)), 2)
    training_set_x = np.array([x[int(random_indices[0])], x[int(random_indices[1])]])
    training_set_y = np.array([y[int(random_indices[0])], y[int(random_indices[1])]])

    mean_squares = []

    for m in hypothesis_m:
        mean_squares.append((m, calc_mean_square(m, training_set_x, training_set_y)))

    mean_squares_arr.append(calc_min_mean_square(mean_squares))
```
[41] ✓ 0.0s

```
for i in range(10000):
    run_trial()
```
[42] ✓ 1m 33.9s

```
a_hat = np.mean(mean_squares_arr)
a_hat
```
[46] ✓ 0.0s

```
0.4586566566566565
```

From the code output above, we see that $\hat{a}$ doesn't match any of the answer choices exactly, so the answer is e

5)

```
x_fine_range = np.linspace(-1, 1, 100000)
bias = np.average((a_hat * x_fine_range - np.sin(np.pi * x_fine_range))**2 )
bias
```

[44] ✓ 0.0s

```
0.28079942557307813
```

The code above returns a bias of 0.2807 which is closest to 0.3, so the answer is b.

6)

```
var_list = []
for a in mean_squares_arr:
    var = np.average((x_fine_range * a - a_hat * x_fine_range) ** 2)
    var_list.append(var)

print(np.average(var_list))
```

[45] ✓ 5.4s

```
0.02415134645276338
```

The code above returns a variance of 0.2807 which is closest to 0.3, so the answer is b.

7) For answer choice b, we know the out-of-sample error to be the bias + variance ≈ 0.28 + 0.02 = 0.30. From the lecture, we know the out-of-sample error are 0.75 & 1.90 respectively. These are both worse than answer choice b. We also only consider linear models to match the linear complexity of our

data resources (2 points in training set only). As such, the answer is b.

8) We take three cases:

$q > N$: Since $q \geq N$,

9-10) Ran out of time for 9 & 10, so I'm going to guess and hedge my answer to both as b or c