CS 156a Problem Set 4

Amitesh Anand Pandey

October 2024

Generalization Error

Problem 1

We know that the generalization bound is

$$\epsilon = \sqrt{\frac{8}{N} \ln \left(\frac{4 m_{\mathcal{H}}(2N)}{\delta} \right)}$$

where ϵ is our tolerance towards the discrepancy in E_{in} and E_{out} and $1 - \delta$ is our confidence in meeting this tolerance. Then we can substitute $m_{\mathcal{H}}(2N) = (2N)^{d_{\mathrm{VC}}} = (2N)^{10} = 2^{10} N^{10}$, $\epsilon = 0.05$, $\delta = 1 - 0.95 = 0.05$, we get

$$0.05 = \sqrt{\frac{8}{N} \ln \left(\frac{4 \cdot 2^{10} N^{10}}{0.05} \right)}$$

Then

$$0.0025N = 8 \ln \left(\frac{4 \cdot 2^{10} N^{10}}{0.05} \right)$$

$$\frac{0.0025}{8} N = \ln \left(\frac{4 \cdot 2^{10} N^{10}}{0.05} \right)$$

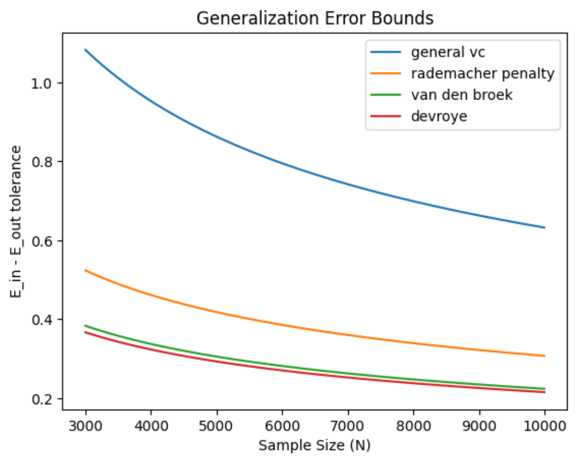
Raising both sides with base power e,

$$(1.00031254883)^N = \frac{4 \cdot 2^{10} N^{10}}{0.05}$$

Using the code We get $N \approx 452957$, thus $N = 460000$, option [**d**] is correct.

Problem 2

So for a hypothesis set $\mathcal{H}$ with $d_{VC} = 50$, setting $\delta = 0.05$, we get the following plots: From this, it is clear that at



$N = 10000$, Devroye bound is the smallest so **[d]** is the correct option, using this code implemented in Python.

Problem 3

For $n = 5$, the Van den Broek bound is the smallest, so **[c]** is the correct option.

Bias and Variance

Problem 4

For this problem, for two points $(x_1, \sin(\pi x_1)), (x_2, \sin(\pi x_2))$, I calculated the mean square error of all hypotheses $g(x) = ax$ for $a = 0.000, 0.001, \ldots 9.999$. Then I picked the a (slope) that minimized the mean square error. I repeated this experiment 10000 times and took the mean slope. I got $\hat{a} \approx 1.42$, so **[e]** is the correct option. Using this code in Python.

Problem 5

For the bias b, we have

$$b = \mathbb{E}_x \left[(\bar{g}(x) - f(x))^2 \right]$$

where $\bar{g}(x) = 1.42x$ and $f(x) = \sin(\pi x)$. Remember that in a real-world problem, an expression for $f(x)$ is not *a priori* available, however this value can still be determined using enough (x, y) pairs out of sample. In our case, we can simply substitute the analytic expression for $f(x)$ since generating these pairs through this function would serve the same purpose. Then

$$b = \frac{1}{2} \int_{-1}^{1} (1.42x - \sin(\pi x))^2 dx = \frac{1}{2} \cdot 0.536 \approx 0.265$$

So **[b]** is the correct option.

Problem 6

For the variance v, we have

$$v = \mathbb{E}_x\left[\mathbb{E}_{\mathcal{D}}\left[(g^{(\mathcal{D})}(x) - \bar{g}(x))^2\right]\right]$$

We already know $\bar{g}(x) = 1.42x$, but $g^{(\mathcal{D})}$ obviously depends on the dataset $\mathcal{D}$. So let's get the best hypotheses $g^{(\mathcal{D}_1)}, g^{(\mathcal{D}_2)}, \dots, g^{(\mathcal{D}_{1000})}$, for 1000 datasets, and then compute the mean value of the variance. On doing so, we get

$$v = \mathbb{E}_x\left[(0.71x)^2\right] = \frac{1}{2}\int_{-1}^{1} 0.53x^2 \mathrm{d}x = \frac{1}{2}\cdot 0.36 = 0.18$$

Thus the correct option is [**a**]. Essentially, we are using two points $(x_1, y_1), (x_2, y_2) \in \mathcal{D}_n$ to generate a best hypothesis $g^{(\mathcal{D}_n)}$, then we're taking the square of its difference with the *average hypothesis* $\bar{g}(x)$, and then repeating this for 1000 $\mathcal{D}$, and taking the expected value of that over the domain $x \in [-1, 1]$. This was the code used.

Problem 7

For $\mathcal{H}$ of the form $h(x) = b$ and $h(x) = ax + b$, we know from lecture that the out of sample error (sum of bias and variance) exceeds 0.5 (which is the out of sample error for $\mathcal{H}$ of the form $h(x) = ax$), we know this from our solutions to the previous two problems. First, let's get the out of sample error for hypotheses of the form $h(x) = ax^2$. Again, let's define $\bar{g}(x) = \hat{a}x^2$ as the average hypothesis. Now we approximate $\bar{g}(x)$ in a similar fashion to as we did in problem 4. On doing so, we get $\hat{a} \approx 1.33$. Thus

$$\bar{g}(x) = 1.33x^2$$

For bias b we have

$$b = \frac{1}{2}\int_{-1}^{1}(1.33x^2 - \sin(\pi x))^2 \mathrm{d}x = \frac{1}{2}\cdot 1.7 = 0.85$$

Since the bias alone is higher, we can rule out hypothesis set $\mathcal{H}$ that contains hypotheses of the form $h(x) = ax^2$. For the hypothesis set that contains hypotheses of the form $h(x) = ax^2 + b$, we will have the average hypothesis

$$\bar{g}(x) = \hat{a}x^2 + \hat{b}$$

Let's approximate $\hat{a}, \hat{b}$ as we have been doing. We get

$$\bar{g}(x) = 1.49x^2 - 0.26$$

For bias b, we have

$$b = \frac{1}{2}\int_{-1}^{1}(1.49x^2 - 0.26 - \sin(\pi x))^2 \mathrm{d}x = \frac{1}{2}\cdot 1.5 = 0.75$$

which alone is larger than our error for hypotheses of the form $h(x) = ax$ thus the correct option is **[b]**. The code used for approximating the average hypothesis was this.

VC-dimension

Problem 8

Remember that if the VC-dimension of a hypothesis set $\mathcal{H}$ is d_{VC}, then the growth function $m_{\mathcal{H}}(N)$ grows as 2^N for as long as the size of the dataset is less than or equal to d_{VC}. In our case, we have

$$m_{\mathcal{H}}(q-1+1) = m_{\mathcal{H}}(q) = 2m_{\mathcal{H}}(q-1) - \binom{q-1}{q} = 2m_{\mathcal{H}}(q-1) = 2^q$$

On the other hand, let $N = q$, we have

$$m_{\mathcal{H}}(q+1) = 2m_{\mathcal{H}}(q) - \binom{q}{q} = 2^{q+1} - 1$$

This means for any dataset of size $q+1$, there is always at least one dichotomy that all hypotheses from $\mathcal{H}$ fail to encompass. Thus $q+1$ is the breaking point, and $q = d_{\text{VC}}$. So the correct option is **[c]**.

Problem 9

First note that it is possible that the intersection of the different hypotheses sets is empty, so on the left side of the inequality we must have 0. When we're taking the intersection of the hypotheses sets, we are *at best* including some hypothesis set $\mathcal{H}_k$ that is contained entirely in all other hypothesis sets. Then, this $\mathcal{H}_k$ *could* be the one with the lowest d_{VC} of all sets $\mathcal{H}_1, \mathcal{H}_2, \ldots \mathcal{H}_K$. Thus **[b]** is the correct option.

Problem 10

Note that when we're taking the union of all hypothesis sets, we are including the hypothesis set $\mathcal{H}_k$ with the highest $d_{\text{VC}}(\mathcal{H}_k$ in its entirety, so the VC dimension of the union will be at least as big as the largest VC dimension of the sets $\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_k$. Now it is conceivable that if the hypothesis sets were individually failing on $\sum d_{VC} + 1$, we could *pick and choose* the hypotheses $h(x)$ from different $\mathcal{H}$ to achieve an improvement. In other words, although $\sum_{k=1}^{K} d_{\text{VC}}(\mathcal{H}_k)$ is tighter, it need not hold if we are *always* able to pick a hypothesis from a unique hypothesis set that shatters the size $d_{\text{max}} + 1$ dataset. Thus **[e]** is the correct option.

Appendix

0.1 Numerical Solution for N

MATLAB Code:

```
syms f(x)
f(x) = (1.00031254883)^x - (80)*(2^10)*(x^10);
sol = vpasolve(f,x,400000)
```

0.2 Plots for Generalization Bounds

```python
import matplotlib.pyplot as plt
import math
def vcbound(N):
    return ((8/N)*math.log((4*(2**50)*(N**50))/0.05))**0.5

def rdbound(N):
    return ((2*math.log(2*(N**51)))/(N))**0.5 + ((2/N)*(math.log((1)/(0.05))))
       *0.5 + 1/N

def vdbound(N):
    return (2 + (4 + 4*N*math.log(120*(2**50)*(N**50)))**0.5)/(2*N)

def dvbound(N):
    return (2 + (4 + 2*(N-2)*math.log(80*(N**100)))**0.5)/(2*(N-2))
x = []
y1 = []
y2 = []
y3 = []
y4 = []
for i in range(3000,10000):
    x.append(i)
    y1.append(vcbound(i))
    y2.append(rdbound(i))
    y3.append(vdbound(i))
    y4.append(dvbound(i))

plt.plot(x,y1, label = 'general vc')
plt.plot(x,y2, label = 'rademacher penalty')
plt.plot(x,y3, label = 'van den broek')
plt.plot(x,y4, label = 'devroye')




plt.xlabel('Sample Size (N)')
plt.ylabel('E_in - E_out tolerance')
plt.title('Generalization Error Bounds')

plt.legend()
```

0.3 Experiments for *a*

```
import random
def sample():
    p1 = random.uniform(-1,1)
    p2 = random.uniform(-1,1)
    return [p1,p2]

def mse(actual_vals, hypo_vals):
    n = len(actual_vals)
    error_sum = 0
    for i in range(n):
        error_sum += (actual_vals[i] - hypo_vals[i])**2
    return error_sum/n

a_vals = []
def minimize(actual_points):
    x1 = actual_points[0]
    x2 = actual_points[1]
    y1 = math.sin(math.pi*x1)
    y2 = math.sin(math.pi*x2)
    slopes = []
    slope_errors = []
    for s in range(1, 100):
        slope = s/10
        hypo_vals = [(slope)*actual_points[0], (slope)*actual_points[1]]
        slopes.append(slope)
        slope_errors.append(mse([y1,y2], hypo_vals))

    return slopes[slope_errors.index(min(slope_errors))]

for i in range(10000):
    a_vals.append(minimize(sample()))

print(sum(a_vals)/len(a_vals))
```

0.4 Approximating Variance

```
sum_diffs = 0
for i in range(1000):
    sum_diffs += abs(a_vals[i] - 1.42)
print(sum_diffs/1000)
```

0.5 Approximating Average Hypotheses

```
a_vals = []
def minimize2(actual_points):
    x1 = actual_points[0]
    x2 = actual_points[1]
    y1 = math.sin(math.pi*x1)
    y2 = math.sin(math.pi*x2)
    slopes = []
    slope_errors = []
    for s in range(1, 100):
        slope = s/10
        hypo_vals = [(slope)*(actual_points[0]**2), (slope)*(actual_points
            [1]**2)]
        slopes.append(slope)
        slope_errors.append(mse([y1,y2], hypo_vals))
    return slopes[slope_errors.index(min(slope_errors))]

for i in range(10000):
    a_vals.append(minimize2(sample()))
print(sum(a_vals)/len(a_vals))
def minimize3(actual_points):
    x1 = actual_points[0]
    x2 = actual_points[1]
    y1 = math.sin(math.pi*x1)
    y2 = math.sin(math.pi*x2)
    hypothesis = []
    slope_errors = []
    for s in range(1, 100):
        slope = s/10
        for k in range(-10, 10):
            b = k/10
            hypo_vals = [(slope)*(actual_points[0]**2) + b, (slope)*(
                actual_points[1]**2) + b]
            hypothesis.append([slope, b])
            slope_errors.append(mse([y1,y2], hypo_vals))
    return hypothesis[slope_errors.index(min(slope_errors))]
sumslope, sumintercept = [0,0]
for i in range(10000):
    slope, intercept = minimize3(sample())
    sumslope += slope
    sumintercept += intercept
print([sumslope/10000, sumintercept/10000])
```