HW2 for Machine Learning

Hsiu Hsuan Yeh

13th April 2023

1 Multiple Choice

1.1 1. (d)

Consider two simple case: $m_H(1) = 2, m_H(2) = 4$, only (d) satisfy.

1.2 2. (a)

The maximum dicatomy is 1126 and by definition, the upper bound:
when $N = d_{vc} + 1$, $1126 < 2^N = 2^{d_{vc}+1}$

$$log_2 1126 < d_{vc} + 1 \Rightarrow log_2 1126 - 1 < d_{vc} < log_2 1126$$

1.3 3. (c)

- (a) can't classified the case +-+-+ so it's VC dimension <= 4
- (b) VC dimension is 4
- (c) can't classified +-+- so it's VC dimension <= 3
- unsure about (d), (c) may not be finite but the VC dimension should > 3

1.4 4. (b)

Consider the following case. If we need $2 * 2$ parameters to fulfill $a_n <= \max_i x_i^T x_i <= b_n$ and $a_o <= \min_i x_i^T x_i <= b_o$ where $a_n = \max_m a_m, a_o = \min_m a_m, b_n = \max_m b_m, b_o = \min_m b_m$. Then, h can shatter if the remained number of x is $<= 2 * (M - 1)$. Since the number of remained parameters is exactly $2 * (M - 1)$. In anoter word, h can'te shatter any $[2 * (M - 1) + 1] + 2 = 2M + 1$ inputs. $d_{vc} <= 2M$

As mentioned above, $[2 * (M - 1)] + 2 = 2M$ inputs can always be shatterd. Nevertheless, $d_{vc} >= 2M$

In Conclusion, the VC dimension is 2M.

1.5 5.(b)

$$d_{vc}(H) <= d \Rightarrow \text{minimum break point} <= d+1$$

Since the definition of the growth function is the maximum dichotomy for some size of data and for N = d + 1, N inputs will always fail to be shattered. Moreover, the condition when N <= d is uncertain. The following two conditions are correct.

- some set of d + 1 distinct inputs is not shattered by H
- any set of d + 1 distinct inputs is not shattered by H

1.6 6. (b)

$$\frac{\partial}{\partial w}\frac{1}{N}\Sigma_{n=1}^{N}(w^T x_n - y_n)^2 = 0$$
$$\Rightarrow w = \frac{\Sigma_{n=1}^{N} y_n x_n}{\Sigma_{n=1}^{N} x_n^2}$$

1.7 7. (c)

log-likelihood:

$$\Sigma_i log\frac{1}{2}\exp(-|x_i - \mu|)$$
$$\frac{\partial}{\partial}\Sigma_i log\frac{1}{2}\exp(-|x_i - \mu|) = \Sigma_i \frac{|x_i - \mu|}{x_i - \mu} = \Sigma_i sign(x_i) = 0$$

To let the equation satisfy, we need the equal amount of -1 and 1. So $\hat{\mu}$ is the median of x_i

1.8 8. (a)

$$\tilde{E}_{in}(w) = \frac{-1}{N} log\Pi_n \tilde{h}(y_n x_n)$$
$$\tilde{E}_{in}(w) = \frac{-1}{N}\Sigma_n log\frac{1 + y_n w^T x_n + |y_n w^T x_n|}{2 + 2|y_n w^T x_n|}$$

Let's first denote $\frac{\partial}{\partial w}|y_n w^T x_n|$ as M

$$\frac{\partial}{\partial w} 1 + y_n w^T x_n + |y_n w^T x_n| = y_n x_n + M$$
$$\frac{\partial}{\partial w}\frac{1}{2 + 2|y_n w^T x_n|} = \frac{-2M}{(2 + 2|y_n w^T x_n|)^2}$$
$$\frac{\partial}{\partial w}\tilde{E}_{in}(w) = \frac{-1}{N}\Sigma_n((\frac{2 + 2|y_n w^T x_n|}{1 + y_n w^T x_n + |y_n w^t x_n|})*(\frac{y_n x_n + M}{2 + 2|y_n w^T x_n|} + \frac{-2M(1 + y_n w^T x_n + |y_n w^T x_n|)}{(2 + 2|y_n w^T x_n|)^2}))$$
$$= \frac{-1}{N}\Sigma_n(\frac{y_n x_n + M}{1 + y_n w^T x_n + |y_n w^t x_n|} + \frac{-M}{1 + |y_n w^T x_n|})$$

After simplify, and no metter the sign of M:

$$\frac{\partial}{\partial w}\tilde{E}_{in}(w) = \frac{-1}{N}\Sigma_n(\frac{y_n x_n}{(1 + y_n w^T x_n + |y_n w^T x_n|)(1 + |y_n x_n|)})$$

1.9 9. (b)

$$\nabla E_{in}(w) = \frac{2}{N}(X^T X w - X^T y) = \frac{2}{N}((X^T X)^T w - X^T y)$$
$$\Rightarrow \nabla^2 E_{in}(w) = \frac{2}{N} X^T X$$

1.10 10. (a)

$$u = -(\frac{2}{N}X^T X)^{-1}\frac{2}{N}(X^T X w_0 - X^T y) = -w_0 + (X^T X)^{-1} X^T y$$
$$w_1 = w_0 + u = (X^T X)^{-1} X^T y$$

Notice that w_{t+1} is the OLS estimator, so it take one step to reach the global minimum.

1.11 11. (d)

$$\mathbb{P}(|E_{in} - E\_out| > 0.05) <= 4 * (2N)^2 * exp\frac{-1}{8}0.05^2 f N$$

- N = 100: $\delta \approx 155077.31751621506$
- N = 1000: $\delta \approx 1.1705850063146269e7$
- N = 10000: $\delta \approx 7.029909379745184e7$
- N = 100000: $\delta \approx 0.004289606188450854$

1.12 12. (d)

If $\tau = 0$, which is noiyless, $E_{out}(w) = min(|\theta|, 0.5) * 1$. While if noisy, the portion of $min(|\theta|, 0.5)$ has the probability (1-τ) being classified correctly. Moreover, the portion of $1 - min(|\theta|, 0.5)$ has the probability τ being classified wrongly. Hence the outsample error:

$$min(|\theta|, 0.5) * (1 - \tau) + (1 - (min(|\theta|, 0.5)) * \tau = min(|\theta|, 0.5) * (1 - 2\tau) + \tau$$

2 Coding

2.1 13. (b)

```
@printf "mean(E_out - E_in): %.5f" test(k=2, τ=0.)
```

```
mean(E_out - E_in): 0.28654
```

2.2 14. (b)

```
@printf "mean(E_out - E_in): %.5f" test(k=128, τ=0.)
```

```
mean(E_out - E_in): 0.00384
```

2.3 15. (c)

```
@printf "mean(E_out - E_in): %.5f" test(k=2, τ=0.2)
```

```
mean(E_out - E_in): 0.42604
```

2.4 16. (b)

```
@printf "mean(E_out - E_in): %.5f" test(k=128, τ=0.2)
```

```
mean(E_out - E_in): 0.01453
```

2.5 17. (c)

```
train_x, train_y = read_data("../train.txt")
test_x, test_y = read_data("../test.txt")

models = fit(train_x, train_y)
i = argmin(insample_error.(models))
best_model = models[i]

println("best of best")
@printf "E_in: %.5f" insample_error(best_model)
```

```
best of best
E_in: 0.02604
```

2.6 18. (e)

```
println("best of best")
@printf "E_out: %.5f" best_model(test_x, test_y, i)
```

```
best of best
E_out: 0.07812
```

2.7 19. (d)

```
ib = argmax(insample_error.(models))
worst_model = models[ib]

println("difference between best of best and worst of best")
@printf "E_in: %.5f" insample_error(worst_model)-insample_error(best_model)
```

```
difference between best of best and worst of best
E_in: 0.30208
```

2.8 20. (b)

```
println("difference between best of best and worst of best")
@printf "E_out: %.5f" worst_model(test_x, test_y, ib)-best_model(test_x, test_y, i)
```

```
difference between best of best and worst of best
E_out: 0.34375
```

3 Code Reference

```julia
module DecisionStump
export simulate_xy, insample_error, fit, test, read_data

using Distributions
import DelimitedFiles: readdlm


struct model
    s::Float64
    θ::Float64
    E_in::Float64
end
direction(a::model) = getproperty(a, :s)
stump(a::model) = getproperty(a, :θ)


check_sign(a) = a == 0. ? -1. : sign(a)


# return scalar
predict(s::Real, x::Real, θ::Real) = s * check_sign(x - θ)
# dim: length(x) * 1
predict(s::Real, x::AbstractVector, θ::Real) = predict.(Ref(s), x, Ref(θ))
# dim: length(x) * length(θ)
predict(s::Real, x::AbstractVector, θ::Vector) = reduce(
    hcat,
    predict.(Ref(s), Ref(x), θ)
)
# dim: length(x) * length(s)
predict(s::Vector, x::AbstractVector, θ::Real) = reduce(
    hcat,
    predict.(s, Ref(x), Ref(θ))
)


function (a::model)(x::AbstractVector, y::Vector)
    pred = predict(direction(a), x, stump(a))
    error = mean(pred .!= y)

    return error
end
(a::model)(x::AbstractMatrix, y::Vector, i) = a(x[:, i], y)


function simulate_xy(size; θ=0., τ=0., s=1.)
    x = rand(Uniform(-0.5, 0.5), size)
    y = predict(s, x, θ)  # dim: size*1

    idx = findall(
        x->x==1,
        rand(Binomial(1, τ), size)
    )
    y[idx] = -1. * y[idx]  # flip

    return x, y
end
```

```julia
function insample_error(x::Vector, y::Vector, θ::Real)
    pred = predict([-1., 1.], x, θ)  # dim: length(x)*2
    error = [mean(i .!= y) for i=eachcol(pred)]

    return error  # dim: 2*1
end
# dim: 2*length(θ)
insample_error(x::Vector, y::Vector, θ::Vector) = reduce(
    hcat,
    insample_error.(Ref(x), Ref(y), θ)
)
insample_error(a::model) = getproperty(a, :E_in)


outsample_error(s, θ, τ) =
    s == 1. ?
    minimum([abs(θ), 0.5])*(1-2τ)+τ : (1-minimum([abs(θ), 0.5]))*(1-2τ)+τ


find_s(idx) = isodd(idx) ? -1. : 1.


function fit(_x::AbstractVector, _y::Vector)
    indices = sortperm(_x)
    x, y, N = _x[indices], _y[indices], length(_x)

    temp1 = zeros(N+1); temp1[2:end] = x
    temp2 = zeros(N+1); temp2[1:end-1] = x
    θ = ((temp1+temp2) ./ 2)[begin+1:end-1]
    push!(θ, -Inf)

    E_in = insample_error(x, y, θ)  # dim: 2*N
    indices = findall(x->x==minimum(E_in), E_in)
    idx = indices[
        argmin([
            find_s(i[1]) * θ[i[2]]
            for i in indices
        ])
    ]

    res = model(
        find_s(idx[1]),
        θ[idx[2]],
        E_in[idx]
    )

    return res
end
fit(_x::Matrix, _y::Vector) = fit.(eachcol(_x), Ref(_y))
```

```julia
function test(n=10000; k, τ)
    res = zeros(n)
    for i = eachindex(res)
        hypothesis = fit(simulate_xy(k; τ=τ)...)
        E_in = insample_error(hypothesis)
        # x_test, y_test = simulate_xy(100000)
        # E_out = hypothesis(x_test, y_test)
        E_out = outsample_error(
            direction(hypothesis),
            stump(hypothesis),
            τ
        )
        res[i] = E_out - E_in
    end

    return mean(res)
end


function read_data(path)
    data = readdlm(path, '\t', Float64, '\n')
    features = data[:, begin:end-1]
    label = data[:, end]

    return features, label
end


end  # end of module
```