HW4 for Machine Learning

Hsiu Hsuan Yeh

11th May 2023

1 Multiple Choice

1.1 1. (a)

$$\frac{\partial}{\partial w}(\frac{1}{N}\Sigma_{n=1}^{N}(wx_n - y_n)^2 + \frac{\lambda}{N}w^2) = \frac{1}{N}\Sigma_{n=1}^{N}(2(wx_n - y_n)x_n) + \frac{2\lambda w}{N}$$

$$\text{by FOC, } \frac{1}{N}\Sigma_{n=1}^{N}(2(wx_n - y_n)x_n) + \frac{2\lambda w}{N} = 0 \Rightarrow C = w*^2 = (\frac{\Sigma_{n=1}^{N}x_n y_n}{\Sigma_{n=1}^{N}x_n^2 + \lambda})^2$$

1.2 2. (b)

$$\Phi(x) = \Gamma^{-1}x \tag{1}$$

$$\frac{\partial}{\partial \tilde{w}}\frac{1}{N}\Sigma_{n=1}^{N}(\tilde{w}\Phi(x_n) - y_n)^2 = \frac{2}{N}(\Gamma^{-1}X^TX\Gamma^{-1}\tilde{w} - \Gamma^{-1}X^Ty) \tag{2}$$

$$\frac{\partial}{\partial \tilde{w}}\frac{\lambda}{N}\tilde{w}^T\tilde{w} = \frac{2\lambda}{N}\tilde{w} \tag{3}$$

$$\text{by FOC, } \frac{2}{N}(\Gamma^{-1}X^TX\Gamma^{-1}\tilde{w} - \Gamma^{-1}X^Ty) + \frac{2\lambda}{N}\tilde{w} = 0 \tag{4}$$

$$\Rightarrow (X^TX(\Gamma^{-1}\tilde{w}) - X^Ty) + \lambda\Gamma\tilde{w} = 0 \tag{5}$$

$$\frac{\partial}{\partial w}(\frac{1}{N}\Sigma_{n=1}^{N}(wx_n - y_n)^2 + \frac{\lambda}{N}\Omega(w)) = \frac{2}{N}(X^TXw - X^Ty) + \frac{\lambda}{N}\Omega'(w) \tag{6}$$

$$\text{by FOC, } \frac{2}{N}(X^TXw - X^Ty) + \frac{\lambda}{N}\Omega'(w) = 0 \tag{7}$$

$$\Rightarrow (X^TXw - X^Ty) + \frac{\lambda}{2}\Omega'(w) = 0 \tag{8}$$

$$\begin{cases} (X^TX(\Gamma^{-1}\tilde{w}) - X^Ty) + \lambda\Gamma\tilde{w} = 0 \\ (X^TXw - X^Ty) + \frac{\lambda}{2}\Omega'(w) = 0 \end{cases} \Rightarrow \begin{cases} w = \Gamma^{-1}\tilde{w} \\ \Omega(w) = w^T\Gamma^2 w \end{cases}$$

1.3 3. (a)

$$err(w,x,+1) = \log(1+\exp(-w^T x)) = \log h(x)^{-1} \tag{9}$$
$$err(w,x,-1) = \log(1+\exp(w^T x)) = \log h(-x)^{-1} \tag{10}$$

$$err_{\text{smooth}}(w,x,+1) = (1-\frac{\alpha}{2})\log h(x)^{-1} + \frac{\alpha}{2}\log h(-x)^{-1} \tag{11}$$
$$err_{\text{smooth}}(w,x,-1) = (1-\frac{\alpha}{2})\log h(-x)^{-1} + \frac{\alpha}{2}\log h(x)^{-1} \tag{12}$$

In the following derivation steps, I just take the positive example $err_{\text{smooth}}(w,x,+1)$ as the demostration. The negative example will have the similar result.

$$\begin{aligned}
err_{\text{smooth}}(w,x,+1) &= (1-\frac{\alpha}{2})\log h(x)^{-1} + \frac{\alpha}{2}\log h(-x)^{-1} \\
&= \log h(x)^{-1} + \frac{\alpha}{2}(\log h(-x)^{-1} - \log h(x)^{-1}) \\
&= err(w,x,+1) + \frac{\alpha}{2}((\log\frac{1}{2} + \log h(-x)^{-1}) - (\log\frac{1}{2} + \log h(x)^{-1})) \\
&= err(w,x,+1) + \alpha(\frac{1}{2}\log\frac{1}{2}h(-x)^{-1} - \frac{1}{2}\log\frac{1}{2}h(x)^{-1}) \\
&= err(w,x,+1) + \alpha(D_{LK}(P_u||P_h) - \log\frac{1}{2}h(x)^{-1}) \\
&= err(w,x,+1) + \alpha D_{LK}(P_u||P_h) - \alpha err(w,x,+1) - \alpha\log\frac{1}{2} \\
&= (1-\alpha)err(w,x,+1) + \alpha D_{LK}(P_u||P_h) - \alpha\log\frac{1}{2}
\end{aligned}$$

$$\text{by FOC, } \frac{\partial}{\partial w}err_{\text{smooth}}(w,x,+1) = 0,$$

$$\Rightarrow err(w,x,+1) + \frac{\alpha}{1-\alpha}D_{LK}(P_u||P_h) = err(w,x,+1) + \lambda D_{LK}(P_u||P_h) = 0, \Rightarrow \Omega(w,x) = D_{LK}(P_u||P_h)$$

1.4 4. (b)

$$\begin{aligned}
E_{loocv} &= \frac{1}{3}((\frac{y_1+y_2}{2} - 1)^2 + (\frac{y_1+1}{2} - y_2)^2 + (\frac{y_2+1}{2} - y_1)^2) \\
&= \frac{1}{2}(y_1^2 - y_1 y_2 + y_2^2 - y_1 - y_2 + 1)
\end{aligned}$$

$$\mathbb{P}(E_{loocv} <= \frac{1}{3}) = \mathbb{P}(y_1^2 - y_1 y_2 + y_2^2 - y_1 - y_2 + \frac{1}{3} <= 0)$$

Define some variables representing the coefficient of the second order polynomial mentioned above and some others used to calculate the area of the revolutioned ellipse:

- coefficient of y_1^2: $a = 1$
- coefficient of $y_1 y_2$: $b = -1$
- coefficient of y_2^2: $c = 1$
- coefficient of y_1: $d = -1$
- coefficient of y_2: $e = -1$
- coefficient of constant: f=1/3

$$m = \frac{2cd - be}{b^2 - 4ac} = 1 \tag{13}$$

$$n = \frac{2ae - bd}{b^2 - 4ac} = 1 \tag{14}$$

$$k = \frac{1}{am^2 + bmn + cn^2 - f} = \frac{3}{2} \tag{15}$$

$$\Rightarrow \mathbb{P}(E_{loocv} <= \frac{1}{3}) = \frac{\frac{2\pi}{k\sqrt{4ac-b^2}}}{4} = \frac{\pi}{3\sqrt{3}}$$

1.5 5. (d)

$$E_{val}(h) = \frac{1}{K}\Sigma_{i=1}^{K} err(h(x_i), y)$$

Due to the the validation examples are generated through exactyl the data distribution and the procss is i.i.d as well:

$$\begin{aligned}
\text{Variance}(E_{val}(h)) &= \text{Variance}(\frac{1}{K}\Sigma_{i=1}^{K} err(h(x_i), y)) \\
&= \frac{1}{K^2} * K * \text{Variance}(err(h(x), y) \\
&= \frac{1}{K}\text{Variance}(err(h(x), y)
\end{aligned}$$

1.6 6. (d)

Because of balanced examples, the one example left out for validation will always be predicted wrongly. Hence, the $E_{\text{loocv}}(A_{\text{majority}}) = \frac{1}{2N}\Sigma_{i=1}^{2N} 1 = 1$

1.7 7. (c)

Since the hopothesis is $h(x_i) = 1 * sign(x_i - \frac{\min_{\{x_i : y_i = +1\}} x_i + \max_{\{x_i : y_i = -1\}} x_i)}{2})$ and there are only two possible cases which wrong prediction occur in validation that is when validation example $x_i = \min\{x_i : y_i = +1\}$ or $\max\{x_i : y_i = -1\}$. Considering the worse case which in both cases, predictions are failed. The upper bound: $E_{loocv}(h) = \frac{2}{N}$.

1.8 8. (e)

Because SVM is the extension of perceptron model and in 1D case, the perceptron model is so called "decision stump". As for the hypothesis for the decision stump: $h(x_i) = 1 * sign(x_i - \frac{\min_{\{x_i:y_i=+1\}} x_i + \max_{\{x_i:y_i=-1\}} x_i)}{2})$, we know that $\theta = \frac{\min_{\{x_i:y_i=+1\}} x_i + \max_{\{x_i:y_i=-1\}} x_i)}{2} = \frac{x_{M+1}+x_M}{2}$ which imply that the largest margin $= \frac{x_{M+1}-x_M}{2}$.

1.9 9. (e)

Denote $w = [w_1, w_2]$. Our objective is to minimize $\frac{1}{2}w^T w$ which is equivalent to minimize $w_1^2 + w_2^2$. The constraints:

$$4w_2 + b \geq 1 \tag{16}$$
$$2w_1 + b \leq -1126 \tag{17}$$
$$-w_1 + b \geq 1 \tag{18}$$
$$b \geq \tag{19}$$

Solve w_1, b under constraint (17), (18), (19) and solve w_2, b through constraint (16) respectively through linear programming(drawing diagram). Solution is:

$$\begin{cases} w_1 & = \frac{-1127}{2}, \text{intersection of two line: } 2w_1 + b = -1126, b = 1 \\ w_2 & = 0, \text{intersection of two line: } 4w_2 + b = 1, b = 1 \\ b & = 1 \end{cases}$$

1.10 10. (d)

By one of the KKT conditions (primal feasible):

$$y_i(\Sigma_{n=1}^N y_n \exp(-\gamma||x_n - x_i||^2)) \geq 1, \forall i \tag{20}$$
$$\Sigma_{n=1}^N (y_n y_i) \exp(-\gamma||x_n - x_i||^2) \geq 1 \tag{21}$$
$$(N-1)\exp(-\gamma\epsilon^2) \geq \Sigma_{n=1}^N (y_n y_i) \exp(-\gamma||x_n - x_i||^2) \geq 1 \tag{22}$$

The formula $(N-1)\exp(-\gamma\epsilon^2)$, the left part(N-1) side imply there are at least one example wich has different y with y_i to make "classification". While the right part($\exp(-\gamma\epsilon^2)$) is due to the constraint $||x_n - x_i|| >= \epsilon, \forall n \neq i$

$$\Rightarrow (N-1)\exp(-\gamma\epsilon^2) \geq 1, \gamma \leq \frac{log(N-1)}{\epsilon^2}$$

1.11 11. (d)

$$
\begin{aligned}
\sqrt{||\phi(x) - \phi(x')||^2} &= \sqrt{(\phi(x) - \phi(x'))^T(\phi(x) - \phi(x'))} \\
&= \sqrt{2 - 2\exp(-\gamma||x - x'||^2)} \\
&\leq \sqrt{2} \approx 1.414
\end{aligned}
$$

2 Coding

```
_train_x, train_y = read_data("../train.txt")
_test_x, test_y = read_data("../test.txt")
train_x, test_x = polynomial_transform.([_train_x, _test_x], Q=4)

λ = [1e6, 1e3, 1, 1e-3, 1e-6]
```

2.1 12. (d)

```
c = l2_param_transform.(λ)
models_l2 = train.(c; y=train_y, x=train_x)

idx_l2 = argmin(binary_error.(
    models_l2;
    y=test_y, x=test_x
))
@printf "log_1_0(λ) = %0.1f has the lowest outsample error" log(10, λ[idx_l2])
```

```
log_1_0(λ) = 3.0 has the lowest outsample error
```

2.2 13. (c)

```
idx = argmin(binary_error.(
    models_l2;
    y=train_y, x=train_x
))
@printf "log_1_0(λ) = %0.1f has the lowest insample error" log(10, λ[idx])
```

```
log_1_0(λ) = 0.0 has the lowest insample error
```

2.3 14. (d)

```
Random.seed!(1234)
count = zeros(Int64, 5)
for i = 1:256
    local idx = pick_param_val(train_y, train_x; param=c).idx
    count[idx] += 1
end
idx = argmax(count)

@printf "in average, log_1_0(λ) = %0.1f has the lowest validation error" log(10, λ[idx])
```

```
in average, log_1_0(λ) = 3.0 has the lowest validation error
```

2.4 15. (c)

```
Random.seed!(1234)
eout = mean([
    binary_error(
        best_model(pick_param_val(train_y, train_x; param=c));
        y=test_y, x=test_x
    )
    for i = 1:256
])

@printf "averaged Eout: %.5f" eout
```

```
averaged Eout: 0.16778
```

2.5 16. (b)

```
Random.seed!(1234)
eout = Vector{Float64}(undef, 256)
for i = eachindex(eout)
    local idx = pick_param_val(train_y, train_x; param=c).idx
    eout[i] = binary_error(
        train(c[idx]; y=train_y, x=train_x); y=test_y, x=test_x
    )
end
@printf "averaged Eout: %.5f" mean(eout)
```

```
averaged Eout: 0.14893
```

2.6 17. (a)

```
Random.seed!(1234)
average_ecv = mean([
    pick_param_crossval(train_y, train_x; param=c, K=5)[2]
    for i = 1:256
])
@printf "averaged Ecv: %.5f" average_ecv
```

```
averaged Ecv: 0.11865
```

2.7 18. (c)

```
c = l1_param_transform.(λ)
models_l1 = train.(c; y=train_y, x=train_x, s=6)

idx_l1 = argmin(binary_error.(
    models_l1;
    y=test_y, x=test_x
))
@printf "log_1_0(λ) = %0.1f has the lowest outsample error" log(10, λ[idx_l1])
```

```
log_1_0(λ) = 0.0 has the lowest outsample error
```

2.8 19. (e)

```
model_l1 = models_l1[idx_l1]
coef_l1 = get_coefficient(model_l1)

@printf "there are %i components of w which absolute value <= 1e-6" sum(abs.(coef_l1)
.<= 1e-6)
```

```
there are 960 components of w which absolute value <= 1e-6
```

2.9 20. (a)

```
model_l2 = models_l2[idx_l2]
coef_l2 = get_coefficient(model_l2)

@printf "there are %i components of w which absolute value <= 1e-6" sum(abs.(coef_l2)
.<= 1e-6)
```

```
there are 1 components of w which absolute value <= 1e-6
```

3 Code Reference

```julia
using Printf
import DelimitedFiles: readdlm
import PyCall: pyimport
import InvertedIndices: Not
import Combinatorics: with_replacement_combinations
using Distributions, Random


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

    return features, label
end


l2_param_transform(x) = 1 / (2x)
l1_param_transform(x) = 1 / (x)


transform(idx; mat) = broadcast(*, eachcol(mat[:, idx])...)
function polynomial_transform(X; Q)
    d = size(X, 2) - 1
    X_ = Matrix{Float64}(undef, size(X, 1), binomial(Q+d, d))
    X_[:, 1:d+1] = X; X = X[:, begin+1:end]

    start = d + 2
    for q = 2:Q
        idx = collect(with_replacement_combinations(1:d, q))
        terminate = start + length(idx) - 1
        X_[:, start:terminate] = reduce(hcat, transform.(idx; mat=X))
        start = terminate + 1
    end
    return X_
end


const liblinear = pyimport("liblinear.liblinearutil")

function train(c; y, x, s=0)
    param = liblinear.parameter("-s $s -c $c -e 0.000001 -q")
    prob = liblinear.problem(y, x)
    model = liblinear.train(prob, param)

    return model
end


function binary_error(model; y, x)
    _, p_acc, _ = liblinear.predict(y, x, model, "-q")
    err = 1 - (p_acc[1]/100)

    return err
end
```

```julia
struct ValResult{T}
    idx::Int64
    errs::Vector{Float64}
    models::T
end
mini_error(a::ValResult) = a.errs[a.idx]
best_model(a::ValResult) = a.models[a.idx]


function pick_param(param, train_y, train_x, eval_y, eval_x)
    models = train.(param; y=train_y, x=train_x)
    err  = binary_error.(models; y=eval_y, x=eval_x)
    idx = argmin(err)
    @inbounds model = models[idx]

    return ValResult(idx, err, models)
end


function pick_param_val(y, x; param)
    N = size(x, 1)
    pos = sample(1:N, 80, replace=false)
    @inbounds train_x, eval_x = x[Not(pos), :], x[pos, :]
    @inbounds train_y, eval_y = y[Not(pos)], y[pos]
    res = pick_param(param, train_y, train_x, eval_y, eval_x)

    return res
end


function pick_param_crossval(y, x; param, K)
    N = size(x, 1); r = Int64(N/K)
    idx, start = repeat(1:N, outer=2), rand(1:N)
    err = Matrix{Float64}(undef, length(param), K)

    @inbounds for j = axes(err, 2)
        terminate = start + r -1
        pos = idx[start:terminate]
        train_x, eval_x = x[Not(pos), :], x[pos, :]
        train_y, eval_y = y[Not(pos)], y[pos]

        err[:, j] = pick_param(param, train_y, train_x, eval_y, eval_x).errs
        start = terminate + 1
    end
    @inbounds param_ecv = mean(err, dims=2)[:, 1]
    idx = argmin(param_ecv)

    return idx, param_ecv[idx]
end


function get_coefficient(model; Q=4, d=10)
    k = binomial(Q+d, d)
    coef = model.get_decfun_coef.(1:k)

    return coef
end
```